Dynamic Thresholding Search for Minimum Vertex Cover in Massive Sparse Graphs

Yuning Chen [a] and Jin-Kao Hao [b,c,*]

[a]*College of System Engineering, National University of Defense Technology, Changsha, China*

[b]*LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c]*Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*

Engineering Applications of Artificial Intelligence
https://doi.org/10.1016/j.engappai.2019.03.015

Abstract

A number of important applications related to complex network analysis require finding small vertex covers in massive graphs. This paper proposes an effective stochastic local search algorithm called DTS_MVC to fulfill this task. Relying on a fast vertex-based search strategy, DTS_MVC effectively explores the search space by alternating between a thresholding search phase during which the algorithm accepts both improving and non-improving solutions that satisfy a dynamically changing quality threshold, and a conditional improving phase where only improving solutions are accepted. A novel non-parametric operation-prohibiting mechanism is introduced to avoid search cycling. Computational experiments on 86 massive real-world benchmark graphs indicate that DTS_MVC performs remarkably well by discovering 7 improved best known results (new upper bounds). Additional experiments are conducted to shed light on the key ingredients of DTS_MVC.

Keywords: Vertex cover; low-complexity heuristics; massive sparse graphs.

1 Introduction

Given a connected, undirected graph $G = (V, E)$ with vertex set V and edge set E, a vertex cover S of G is a subset of V ($S \subset V$) such that each edge of the graph is incident to at least one vertex of S. The Minimum Vertex Cover (MVC)

* Corresponding author.
Email addresses: cynnudt@hotmail.com (Yuning Chen), jin-kao.hao@univ-angers.fr (Jin-Kao Hao).

problem is to determine a vertex cover of minimum cardinality. MVC is equivalent to the Maximum Independent Set (MIS) problem and the Maximum Clique (MC) problem since given a minimum vertex cover S of G, $V \setminus S$ is a maximum independent set in G and a maximum clique of $\overline{G}$ (the complement of G) [42]. In addition to many conventional applications, MVC as well as its equivalent models become more and more popular in important areas like social network analysis [14], bioinformatics [21] and economics [3, 43].

Solving MVC in the general case is computationally challenging given that its decision version is NP-complete [16]. Due to its NP-hardness, most previous MVC algorithms adopted stochastic local search (SLS) heuristics (e.g., [5–7, 29, 31, 36]) to seek high-quality sub-optimal solutions in reasonable time. Several exact algorithms based on the general Branch and Bound method were also proposed for the equivalent MC problem (e.g., [20, 23, 34, 38]). These previous algorithms were demonstrated to be effective in solving traditional graphs from the DIMACS and BHOSLIB benchmark sets which are of small or medium size.

Meanwhile, the fast growth of the Internet and the recent advances in information technology have generated a large amount of data whose underlying graphs are much larger than the traditional benchmarks. These real-world graphs (like social networks) are typically massive, with tens of millions of vertices, and sparse with low edge densities where the degrees of vertices often follow a power-law distribution [22]. These massive graphs pose serious challenges to existing solution approaches [24]. First, many existing algorithm implementations cannot be applied due to their matrix representation of the graph [39]. With the matrix representation, graphs with millions of vertices simply cannot be loaded into a computer's working memory. Second, many best performing SLS algorithms adopt the best-improvement (or operator-based) search strategy for solution transformations on a scale-correlated candidate set (e.g., [7, 19, 30, 41]) (see Sect. 3.3). Such a strategy incurs a high time complexity of the algorithm per iteration and becomes inefficient and impracticable on massive graphs. Indeed, each application of this strategy in the setting of massive graphs requires millions of evaluation operations, which, even if quite simple, will necessarily lead to a very low computational efficiency and slow down the algorithm.

Developing suitable algorithms for MVC and its equivalent problems on massive graphs is receiving increasing research attention in recent years. In [26, 27, 33, 39], exact algorithms were presented to solve the equivalent MC optimally. On the other hand, due to the size of the massive sparse graphs and the intractability of the considered problems, SLS represents a natural solution approach to find high-quality approximate solutions, which nevertheless, needs to incorporate dedicated designs to be effective. In [1], a GRASP heuristic was introduced for MC which integrates graph decomposition schemes and an external-memory procedure. One notices that applying a MC algorithm to MVC requires operating on a highly dense and massive complement graph, which makes the algorithm inappropriate for solving MVC.

Very recently, a dedicated SLS algorithm called FastVC was introduced to find small vertex colors in massive graphs [4], which was built upon the NuMVC algorithm [7] and includes specific heuristics for initial solution generations and solution transformations. Another new SLS algorithm called LinCom [13] adopts the same algorithm scheme as FastVC. LinCom distinguishes itself from FastVC by integrating reduction rules for solution initialization and employing an efficient data structure to implement the best improvement heuristic. LinCom was shown to be able to compete favorably with FastVC on a number of massive real-world graphs. To our knowledge, FastVC and LinCom are the current best performing heuristic algorithms for the MVC problem considered in this work and will serve as the main references for the assessment of our proposed algorithm.

In addition to the above studies on MVC, we notice some very recent efforts devoted to extensions of the basic MVC and clique models such as the robust clique problem with uncertain edge failures [44], the clique relaxation problems (k-blocks and k-robust 2-clubs, [2]), the degree-based-quasi-clique problem [25], the maximum quasi-clique problem [28], the maximum s-plex problem [15, 45], the maximum balanced biclique problem [40,46], and the maximum induced biclique problem [35]. These extended models together with the classic MVC and clique models provide a useful and relevant means for modeling and analyzing massive graphs and complex networks. Designing new methods able to effectively handle these problems for massive graphs represents a real interest and a formidable challenge as well.

In this work, we focus on solving MVC on massive sparse graphs and introduce a fast and effective SLS heuristic algorithm called DTS_MVC (Dynamic Thresholding Search for MVC). DTS_MVC basically alternates between a *thresholding search phase* where the algorithm accepts new (either improving or non-improving) solutions subject to a dynamically evolving quality threshold and a *conditional improving phase* where only improving solutions are sought. The original features of DTS_MVC and the contributions of this work are identified as follows.

- First, DTS_MVC uses a randomized greedy procedure based on vertex degree information to build its initial solution (vertex cover). With the same complexity ($O(|E|)$) as the conventional random construction procedure, the greedy procedure is able to generate initial solutions of better quality and helps to speed up the search process of DTS_MVC.
- Second, the thresholding search phase employs a vertex-based strategy and a parameter-free operation-prohibiting mechanism for solution transformations. The vertex-based strategy applies the best operator taken from a small operator candidate set to a given vertex (randomly chosen) to transform the incumbent solution. From the computational point of view, this approach is significantly more efficient than the traditional operator-based strategy which needs, for solution transformations, to identify among a large vertex candidate set the best vertex to which the given transformation operator is applied.

- Third, DTS_MVC uses a random 'switch' to control the input of each thresholding search phase which is either the local optimum from the hill-climbing improving phase or the solution returned by the last round of thresholding search. This design naturally reinforces the balance between intensification and diversification of the search process.
- Fourth, computational assessments on 86 massive real-world graphs indicate that DTS_MVC competes favorably with two state-of-the-art MVC solvers dedicated to massive graphs (FastVC and LinCom) by improving 7 and matching 64 best known results.
- Finally, the proposed vertex-based strategy and the operation-prohibiting mechanism are essentially problem-independent and can be advantageously used to design search algorithms for other graph problems. Given that these techniques help to accelerate the search process and to simplify the algorithmic procedure, they will be particularly useful and promising when the problem scale is large and the computing time allowed is limited.

The rest of the paper is organized as follows. After introducing preliminary definitions and notations (Sect. 2), we describe in Section 3 the proposed algorithm and its ingredients. In Section 4, we present computational results and comparisons with state-of-the-art methods and analyze the key algorithmic components. In the last section, we draw concluding remarks and indicate some ideas for future studies.

2 Basic Definitions and Notations

Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, let $N(v) = \{u \in V : (u, v) \in E\}$ denote the neighborhood of vertex $v \in V$. Let $deg(v) = |N(v)|$ and $avgDeg = \sum_{v \in V} deg(v)/n$ denote respectively the degree of a vertex v and the average degree of G. Let S be a solution (i.e., a vertex cover of G), then we defined the following vertex sets which are useful to describe the proposed algorithm.

- The *expanding neighborhood* of a vertex $v \in S$ relative to $V \backslash S$, denoted as $N^E(v)$, is the set of neighbors of v that belong to $V \backslash S$, i.e., $N^E(v) = \{u \in V \backslash S : (u, v) \in E\}$.
- The *mapping neighborhood* of a vertex $v \in V \backslash S$, denoted as $N^M(v)$, is the set of neighbors of v in S whose expanding neighborhood has a single vertex, i.e., $N^M(v) = \{u \in S : (u, v) \in E, |N^E(u)| = 1\}$.
- The *improving neighbor set* of S ($N_I(S)$) is a set of vertices in S that are not connected to any vertex in $V \backslash S$, i.e., $N_I(S) = \{v \in S : |N^E(v)| = 0\}$. By definition, any vertex in $N_I(S)$ can be directly removed from S to obtain a better solution (i.e., a smaller vertex cover).

Figure 1 shows an illustrative example of the last three notations, where the considered graph has 6 vertices with a given vertex cover $S = \{v_1, v_2, v_3, v_4\}$.

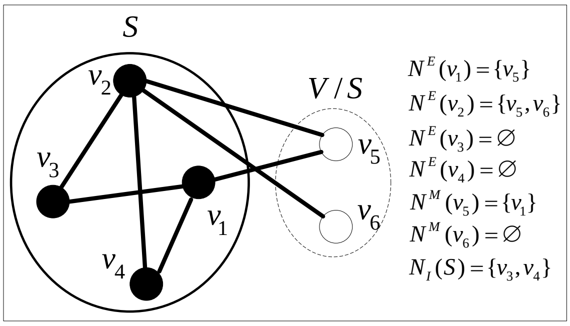

Fig. 1. Illustrative example of the basic notations

3 The DTS_MVC Algorithm

3.1 General Outline

The DTS_MVC algorithm (Alg. 1) proposed in this work starts by constructing an initial vertex cover using the *GreedyConstruct* procedure (Sect. 3.2), and then alternates between a *thresholding search phase* (Sect. 3.3) and a *conditional improving phase* (Sect. 3.4) to find increasingly better solutions (smaller vertex covers).

In the thresholding search phase (lines 8-27 of Alg. 1), the algorithm uses a so-called *vertex-based strategy* to prospect for good solutions by scanning the vertices one by one *in random order* and for each considered vertex, applying accordingly one of three basic transformation or move operators: DROP, ADD and SWAP (see Sect. 3.3 for their definitions). Precisely, after picking a vertex, the algorithm examines two associated operators ({DROP, SWAP} or {SWAP, ADD} depending on whether the vertex is in or out of the current vertex cover) in a specific order (ensuring that the most suitable operator is always applied first), and any resulting neighboring solution that satisfies a quality threshold is accepted immediately. With a responsive mechanism [9], the quality threshold dynamically evolves with the best solution found along the search process. This fast vertex-based strategy helps the algorithm to efficiently visit a large number of solutions for a given search effort. In this phase, the algorithm also uses an operation-prohibiting mechanism (a parameter-free tabu list [17]) to avoid reversing recently performed moves. A round of the thresholding search phase terminates when all vertices in V have been examined once and only once. Thanks to the vertex-based strategy and the operation-prohibiting mechanism, the search process is expected to explore various zones of the search space without being easily trapped in local optima.

As a complement to the thresholding search phase, DTS_MVC performs a strict hill-climbing in the conditional improving phase to intensify the search (lines 28-39 of Alg. 1). The hill-climbing procedure shrinks the current vertex cover S by dropping all vertices that are not connected to any vertex in $V \backslash S$. This phase is named ‘conditional’ since the local optimum of the hill-climbing procedure is accepted to become the starting solution of the next round of thresholding search only when the random switch (‘on’ or ‘off’) is ‘on’.

DTS_MVC iterates the above two phases to seek increasingly better solutions until a prefixed *cutoff* time is reached. In what follows, we present the three main components of the DTS_MVC algorithm: *GreedyConstruct* procedure, the *thresholding search phase* and the *conditional improving phase*.

Algorithm 1: DTS_MVC Algorithm

```
Data: graph G=(V,E); integer δ; the cutoff time;
Result: the smallest vertex cover S* found
initialize age as an all-0 array ;
round := 0;   S := GreedyConstruct();   S* := S;   Record := |S*| ;
while elapsed time < cutoff do
    round := round + 1 ;
    // Thresholding Search Phase
    Randomly shuffle all vertices in V ;
    for each v ∈ V do
        if v ∈ S then
            if |N^E(v)| = 0 then
                S := S\{v} ;
                if |S| < |S*| then
                    S* := S
            else if |N^E(v)| = 1 ∧ age[v] < round then
                u := the only vertex in N^E(v);   S := S\{v};   S := S ∪ {u};
                age[u] := round
        else
            if |N^M(v)| > 0 ∧ age[v] < round then
                u := a random vertex in N^M(v);   S := S\{u};   S := S ∪ {v};
                age[v] := round
            else if |S| + 1 ≤ Record + δ then
                S := S ∪ {v} ;
    // Conditional Improving Phase
    Record := |S*|; S' := S ;
    while N_I(S) ≠ ∅ do
        u := a random vertex in N_I(S);   S := S\{u} ;
    if |S| < |S*| then
        S* := S
    switch := a random value in {0, 1} ;
    if switch = 1 then
        Record := |S*| ;
    else
        S := S' ;
return S*
```

3.2 The GreedyConstruct Procedure

DTS_MVC requires an initial vertex cover as its starting solution. Since random solutions are generally of mediocre quality, using such a solution to start the search will significantly slow down the algorithm for massive graphs. For this reason, we devise a fast greedy procedure (*GreedyConstruct*) which offers a suitable balance between the time complexity and the solution quality.

Algorithm 2: GreedyConstruct Procedure

Data: graph G=(V,E);
Result: a vertex cover S

$S := \emptyset$;
while $N_I(S) \neq \emptyset$ **do**
  if *avgDeg* $< |N_I(S)|$ **then**
    $u :=$ the vertex with the smallest degree among the *avgDeg* vertices which are randomly selected from $N_I(S)$;
  else
    $u :=$ the vertex with the smallest degree in $N_I(S)$;
  $S := S \backslash \{u\}$;
return S

GreedyConstruct constructs an initial vertex cover using the vertex degree as heuristic information which favors exclusion of vertices with a small degree from the current solution. The intuition behind this heuristic is that the vertices with a small degree are less likely to connect to each other and thus potentially allow more such vertices to be excluded from the solution. Starting from a cover S with all vertices of V, *GreedyConstruct* (Alg. 2) iteratively shrinks S by dropping one vertex of the improving neighbor set $N_I(S)$ (Sect. 2) out of S at one time until $N_I(S)$ becomes empty. To decide a vertex u to be excluded from S, the procedure distinguishes two situations depending on the relationship between *avgDeg* and the size of $N_I(S)$. If *avgDeg* is smaller than the cardinality of $N_I(S)$, u is the vertex with the smallest degree among the *avgDeg* vertices randomly selected from $N_I(S)$; otherwise u is the vertex with the smallest degree in $N_I(S)$.

Proposition 1 *Given $G = (V, E)$ with $|V| = n$ and $|E| = m$, the time complexity of the GreedyConstruct procedure to construct a minimal vertex cover is $O(m)$.*

Proof Let $S = \{v_{i_1}, v_{i_2}, ..., v_{i_t}\}$ denote the vertex cover produced by the GreedyConstruct procedure. Since G is a connected graph and S is a vertex cover, it is clear that $t \leq \lceil n/2 \rceil$. To decide v_{i_j}, the algorithm needs to perform at most *avgDeg* comparisons among vertices selected from $N_I(S)$. After the exclusion of v_{i_j} from S, the algorithm needs to update the improving neighbor set $N_I(S)$ by traversing $deg(v_{i_j})$ neighbors of v_{i_j}. Therefore, the complexity of the GreedyConstruct procedure is $O((t - avgDeg) * avgDeg + avgDeg(1 + avgDeg)/2 + \sum_{j=1}^{t} deg(v_{i_j})) < O(t * avgDeg + \sum_{j=1}^{t} deg(v_{i_j})) \approx O(m + m) \approx O(m)$.

Since an efficient implementation of the random construction procedure also requires to maintain an improving neighbor set $N_I(S)$ from which a vertex is randomly selected to shrink the vertex cover at each iteration, the complexity of the random construction is $O(\sum_{j=1}^{t} deg(v_{i_j})) \approx O(m)$, i.e., the same order of magnitude as that of our greedy construction. However, with initial solutions of better quality, our greedy construction generally helps DTS_MVC to achieve better results with negligible additional costs (see computational results and discussions in Sect.4.3).

3.3 The Thresholding Search Phase

In the Thresholding Search Phase (lines 8-27 of Alg. 1), both improving and non-improving solutions can be accepted by a combined use of three different basic operators (DROP, ADD and SWAP). DROP kicks a vertex out of the current vertex cover; ADD expands the current vertex cover by including a new vertex; SWAP exchanges a vertex v of the vertex cover with a vertex u out of the vertex cover. We note that the idea of systematically allowing non-improving solutions to be accepted by reference to a solution quality threshold is featured by the threshold accepting heuristic [12] which has been successfully applied to solve a number of other combinatorial optimization problems (e.g., [8–11,37]). The main innovations in this phase include the vertex-based exploration strategy and the non-parametric operation-prohibiting mechanism whose working principles and their contribution to the solution of the problem are detailed below.

Previous SLS algorithms for MVC and its equivalent problems usually employ an *operator-based* strategy coupled with a best-improvement heuristic [7, 19, 30] which operates as follows. Given the current solution, such an algorithm applies a transformation operator to a best element (vertex) selected from a vertex candidate set whose size is correlated to the number of vertices of G and grows as the problem scale increases. To identify the best vertex from such a vertex candidate set requires numerous comparison operations which is time-consuming. Though the operator-based strategy works well for small and medium graphs (up to several thousands of vertices), it becomes unacceptable for massive data sets.

Unlike these previous approaches, we introduce a low-complexity search mechanism in the thresholding search phase. Instead of seeking a best vertex for an operator, DTS_MVC employs a *vertex-based* strategy to determine, for a given vertex, the best operator to be applied from a set of applicable candidate operators. This strategy is efficient since the candidate operator set is small (only two possible operators for a vertex in our case) and is not related to the problem size. The idea of the vertex-based strategy is in line with the common intuition that, in the context of SLS algorithms for massive problems, it is preferable to perform many relatively simple, but efficiently computable search steps rather than few complex search steps [18]. Thanks to the vertex-based strategy, DTS_MVC is able to visit many different solutions and explore various areas of the search space to locate high quality solutions.

An important issue for SLS algorithms is to prevent the search from revisiting previously visited solutions. To address this issue, we devise a novel non-parameter operation-prohibiting (OP) mechanism in the thresholding search phase. The key idea of this mechanism is to ensure that an operation applicable to a vertex is applied at most once in a round of thresholding search. It is clear that for DROP and ADD, this condition is always satisfied. Therefore, the OP mechanism is used to ensure that the SWAP operation for a pair of vertices (v, u) is applied at most once as well. For this, we introduce an 'AGE' counter for each vertex u such that each time a vertex u is swapped into the solution, its AGE counter is set to be the current *round* number. Then only vertices with an AGE value smaller than the current $round$ number are considered for SWAP operations during this round of thresholding search. The OP mechanism, combined with the random permutation of all vertices before each round of thresholding search (line 7 of Alg. 1), can be viewed as a parameter-free tabu mechanism with a random tabu tenure for each performed operation [17]. Our empirical experience indicates that the random permutation of all vertices is an important operation that greatly boosts the performance of DTS_MVC.

Based on the three basic move operators, the thresholding search phase of DTS_MVC combines the vertex-based strategy and the operation-prohibiting mechanism to make a balanced examination of the search space. For each round of this phase, the algorithm *first randomly shuffles all vertices* in V and then visits the vertices one by one. For each vertex v under consideration, we define its operator candidate set according to whether the vertex is inside or outside the vertex cover S.

Case 1: If v belongs to S, the operator candidate set is composed of DROP and SWAP. DROP is tested before SWAP since DROP increases the solution quality while SWAP keeps it unchanged. If both tests fail, the algorithm simply skips v. DROP is applicable if the expanding neighborhood of v is empty (i.e., v belongs to the improving neighbor set $N_I(S)$, Sect. 2). The SWAP operation is applicable if v simultaneously satisfies two conditions: 1) v is not involved in any applied SWAP operation at the current round; 2) v is connected to exactly one vertex in $V \backslash S$ (i.e., $|N^E(v)| = 1$ defined in Sect. 2, this vertex is the unique vertex that can be swapped with v).

Case 2: If v is out of the vertex cover ($v \in V \backslash S$), the operator candidate set is composed of SWAP and ADD. The applicability of SWAP is tested before ADD since SWAP does not deteriorate the solution quality while DROP does. If both tests fail, the algorithm simply skips v. SWAP is applicable if v simultaneously satisfies two conditions: 1) v was not involved in any applied SWAP operation at the current round; 2) the mapping neighborhood of v ($N^M(v)$, Sect. 2) is not empty. The vertex $u \in S$ to be swapped with v is a random vertex in $N^M(v)$. ADD is applicable if the number of vertices of S after the operation is still below a threshold determined by $Record + \delta$ where $Record$ is the cardinality of the recorded smallest vertex cover and δ (a small positive integer) is a parameter.

3.4 The Conditional Improving Phase

The balance between intensification and diversification is another important issue for SLS algorithms. The thresholding search procedure provides a form of diversification since it accepts both non-improving and deteriorating solutions. To complement this, an intensification procedure is needed to ensure a more focused and directed search. For this purpose, a hill-climbing procedure is employed in the conditional improving phase (lines 28-39 of Alg. 1). The aim of the hill-climbing procedure is to attain a local optimum using the DROP operator starting from the solution S returned by the thresholding search phase. This is achieved by iteratively selecting a random vertex from $N_I(S)$ and kicking it out of the solution until $N_I(S)$ becomes empty. The local optimum of the hill-climbing procedure is interesting since it can update the recorded best solution S^* (lines 33-34, Alg. 1).

However, from a long-term perspective of the search process, a local optimum identified such a technique is not necessarily a good starting point for the next round of the thresholding search phase. For this reason, the conditional improving phase uses a random switch ('on' or 'off') to decide its output solution which will serve as the starting solution of the next round of thresholding search. This mechanism proposed in this work aims to balance the search intensification and diversification. When the switch is 'on', the output solution is the local optimum; otherwise it is the solution returned by the last round of thresholding search. This design is based on the following considerations. Though the thresholding search procedure provides a form of controlled diversification which allows the search to escape some local optima, it may be insufficient to escape deep local optima traps due to the quality limit fixed by the δ value. Instead of trying to escape such traps, the algorithm simply skips them occasionally. On the other hand, since some local optima are easy to escape and other optima are hard to get out of, it could be beneficial to start the next round of thresholding search from the local optimum of the hill-climbing procedure as well.

In our implementation, the conditional improving phase first sets the $Record$ value to the size of the current smallest vertex cover and memorizes in S' the solution returned by the thresholding search phase. It then performs the hill-climbing procedure to attain the local optimum S and uses it to update the recorded best solution S^* if needed. The algorithm accepts the local optimum S and sets $Record$ to the cardinality of the new best vertex cover S^* if the random switch is 'on', or continues with S' otherwise.

4 Computational Experiments

We evaluate the performance of DTS_MVC based on a wide range of 86 massive real-world graphs with respect to two most recent MVC solvers dedicated to massive graphs: FastVC [4] and LinCom [13].

4.1 Benchmark Instances

The 86 tested benchmark instances from the Network Data Repository online [32] are massive real-world graphs (with up to millions of vertices and tens of millions of edges) from 10 general collections, including biological networks, collaboration networks, facebook networks, interaction networks, infrastructure networks, amazon recommend networks, scientific computation networks, social networks, technological networks, and web link networks.

4.2 Experimental Settings and Computational Results

The proposed DTS_MVC algorithm was coded in C++ and compiled by GNU g++ 4.1.2 with the '-O3' flag. The experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Ubuntu 12.04. Solving the DIMACS machine benchmark graphs r300.5, r400.5 and r500.5 [1] on our computer (without any compilation optimization flag) requires 0.40, 2.50 and 9.55 seconds respectively.

The proposed algorithm DTS_MVC as well as the two reference algorithms were run 100 times for each instance with the time limit of 1000 seconds per run (a commonly used stopping condition in the literature [4, 13]). We ran the C++ source codes of FastVC [2] on our machine for the sake of fair comparison. The results of LinCom were extracted directly from the original paper since its source code is not available to us. Notice in this comparison, the stopping condition of 1000 seconds is advantageous for LinCom since according to the Standard Performance Evaluation Corporation (www.spec.org), the machine used to run LinCom [13] is 1.17 times faster than our computer.

To assess the peak performance of DTS_MVC, we conducted the above experiment for multiple values of δ (the unique parameter of DTS_MVC) and reported the best δ value and its corresponding results. Specifically, δ is tuned in an iterated manner. Initialized with a value of 0, δ is incremented by 1 each time and iterated until the

[1] ftp://dimacs.rutgers.edu/pub/dsj/clique/
[2] http://lcs.ios.ac.cn/ caisw/MVC.html

performance of DTS_MVC is not improved for consecutive 2 iterations. As we will see later from the reported results, for most instances (76 out of 86 cases), the best δ takes the value of 1 which means in practice only a limited tuning effort is needed for these instances.

For each algorithm and for each instance, we report the smallest size (f_{best}) and the averaged size (f_{avg}) over the 100 best vertex covers (one for each run) discovered by the algorithm, as well as the averaged running time in seconds (CPU(s)) over 100 runs when the best solution is first encountered by the algorithm in each run (the averaged running times of LinCom are not included in this comparison since they were not reported in the original paper). Due to the difference of the computational environments, the results obtained by running the source codes of FastVC on our machine are not exactly the same as those reported in [4]. For the purpose of a more comprehensive comparison, we also included the best results reported in [4] as the current best known results (BKRs).

The detailed comparative results of DTS_MVC together with the two reference algorithms on the 86 benchmarks are listed in the Appendix (Table A.1 and A.2), while Table 1 shows a summary of these detailed results and provides a general picture of the performance of the compared algorithms. Note that only the results of 59 instances (out of 86) were reported for LinCom in the original paper. From these results, we can make the following observations.

- First, DTS_MVC performs remarkably well on these real-world graphs. Indeed, it matches or improves on the best known results (BKRs) for most instances (71 out of 86 cases). In particular, it is able to find 7 new BKRs (indicated with the '*' symbol in Table A.1 and Table A.2) that were never identified before.
- Second, DTS_MVC competes very favorably with FastVC. Indeed, DTS_MVC finds substantially more BKRs than FastVC (71 vs. 57). In terms of the best solution found, DTS_MVC attains a better outcome for 27 graphs, an equal result for 57 graphs and a worse result for only two cases. As to the average performance (f_{avg}), DTS_MVC performs more stably than FastVC. Indeed, DTS_MVC achieves a better value for 41 graphs, an equal value for 33 instances, and a worse value in 12 cases. These results also show that within the same time limit (1000 seconds), DTS_MVC is able to find improving solutions while FastVC gets stuck early before the time limit.
- Third, compared to LinCom which is one of the most recent and the best performing MVC algorithms dedicated to massive graphs, DTS_MVC remains very competitive even if the improvement of our algorithm over LinCom is relatively small. Table 1 indicates that in terms of the best result (f_{best}), the number of instances where DTS_MVC is better than LinCom is more than where DTS_MVC is worse (i.e., $\#Better$ vs. $\#Worse$ = 15:14). DTS_MVC is more robust than LinCom since DTS_MVC shows a better average performance in 37 instances. This number is larger than the 22 cases where DTS_MVC attains a worse result. Notice that, the results of LinCom were obtained on a computer which is 1.17

time fascter than our computer.

- Fourth, there are two collections of graphs (prefix 'ca-' and 'ia-') where all three compared algorithms perform similarly. Among the other networks, DTS_MVC dominates the other two algorithms on the collection prefixed by 'socfb-' while LinCom outperforms the other algorithms on the collections prefixed by 'inf-' and 'soc-'. This observation shows that DTS_MVC is a good complement to the state-of-the-art MVC solution approaches.

Table 1
Statistical data on comparative results of DTS_MVC with two state-of-the-art algorithms (FastVC [4] and LinCom [13]). For a given algorithm pair, #Instance represents the number of instances in comparison. For each algorithm pair and each indicator, #Better, #Equal and #Worse count the number of instances on which the first algorithm achieves a value that is better, equal to or worse than the second algorithm respectively.

Algorithm pair	#Instance	Indicator	#Better	#Equal	#Worse
DTS_MVC vs. FastVC	86	f_{best}	27	57	2
		f_{avg}	41	33	12
DTS_MVC vs. LinCom	59	f_{best}	15	30	14
		f_{avg}	37	0	22

4.3 Discussion

A second experiment was carried out to investigate the effectiveness of four key ingredients of DTS_MVC: the initialization procedure, the switch condition of the improving phase, the random shuffling strategy of vertices before each thresholding search phase, and the operation-prohibiting mechanism. To this end, we studied four algorithm variants, each with one or two ingredients removed from the original DTS_MVC algorithm:

- DTS_MVC$_1$: with the greedy initialization procedure replaced by a pure random initialization procedure;
- DTS_MVC$_2$: with the switch condition removed;
- DTS_MVC$_3$: with the random shuffling strategy disabled;
- DTS_MVC$_4$: with both random shuffling strategy and the operation-prohibiting mechanism disabled.

We tested these algorithm variants on the 86 benchmark networks under the same experimental protocol as DTS_MVC. The pairwise comparative results of DTS_MVC with each of its variants (or between two algorithm variants) were summarized in Table 2. To give a general picture of the performances of the four algorithm variants with respect to DTS_MVC, we also provided two plots of their best and average results in Figure 2 and 3. The horizontal axis of these plots shows the instance serial number that indicates the order in which the instance appears in Table A.1 and A.2. The vertical axis shows the performance gap (either best or average) in percentage, calculated as $(f - f_{dts}) \times 100/f_{dts}$ where f is the (best or average) solution value of the algorithm variant and f_{dts} is the (best or average) solution value of DTS_MVC.

Table 2
Statistical comparative results of DTS_MVC with its four algorithm variants over 86 benchmark instances. For each algorithm pair and each indicator, #Better, #Equal and #Worse count the number of instances on which the first algorithm achieves a value that is better, equal or worse than the second algorithm respectively.

Algorithm Pair	Indicator	#Better	#Equal	#Worse
DTS_MVC vs. DTS_MVC$_1$	f_{best}	25	58	3
	f_{avg}	42	39	5
DTS_MVC vs. DTS_MVC$_2$	f_{best}	28	55	3
	f_{avg}	36	41	9
DTS_MVC vs. DTS_MVC$_3$	f_{best}	64	22	0
	f_{avg}	75	11	0
DTS_MVC$_3$ vs. DTS_MVC$_4$	f_{best}	69	17	0
	f_{avg}	77	9	0

From Table 2 and Figures A.1-A.2, we can make the following observations:

- DTS_MVC performs better than DTS_MVC$_1$ in terms of both best results (more #Better than #Worse: 25 vs. 3) and average results (more #Better than #Worse: 42 vs. 5). For 39 instances where both DTS_MVC and DTS_MVC$_1$ consistently attain the BKR, DTS_MVC is faster than DTS_MVC$_1$ on average (42.60s vs. 56.39s). This observation shows that compared to the random construction procedure, our greedy construction procedure provides better initial solutions to promote the overall performance of DTS_MVC, and thanks to these high quality starting points, DTS_MVC arrives at a still better solution with generally less computing time than DTS_MVC$_1$.
- DTS_MVC also outperforms DTS_MVC$_2$ in terms of both best results (more #Better than #Worse: 28 vs. 3) and average results (more #Better than #Worse: 36 vs. 9). This confirms the effectiveness of the random ‘switch’ condition which contributes to the balance of intensification and diversification of DTS_MVC.
- When comparing DTS_MVC with DTS_MVC$_3$, we observe a more significant dominance of DTS_MVC (see Figure A.1 and A.2, the black line of DTS_MVC is mostly below the the red line of DTS_MVC$_3$). Indeed, DTS_MVC is always better than or equal to DTS_MVC$_3$ in terms of both best result and average results. In particular, DTS_MVC achieves a better f_{best} result for 64 out of 86 instances (74.4%) and a better f_{avg} result for 75 out of 86 instances (87.2%). This observation demonstrates the critical role of the random shuffling strategy in boosting the performance of DTS_MVC.
- Although much worse than DTS_MVC, DTS_MVC$_3$ clearly dominates DTS_MVC$_4$ thanks to the inclusion of the operation-prohibiting mechanism (see Figure A.1 and A.2, the red line of DTS_MVC$_3$ is generally below the the green line of DTS_MVC$_4$). Indeed, DTS_MVC$_3$ is consistently better than (for more than 80% of all tested instances) or equal to DTS_MVC$_4$ in terms of both best and average results. This observation confirms the effectiveness of our operation-prohibiting mechanism in avoiding the search cycling issue.

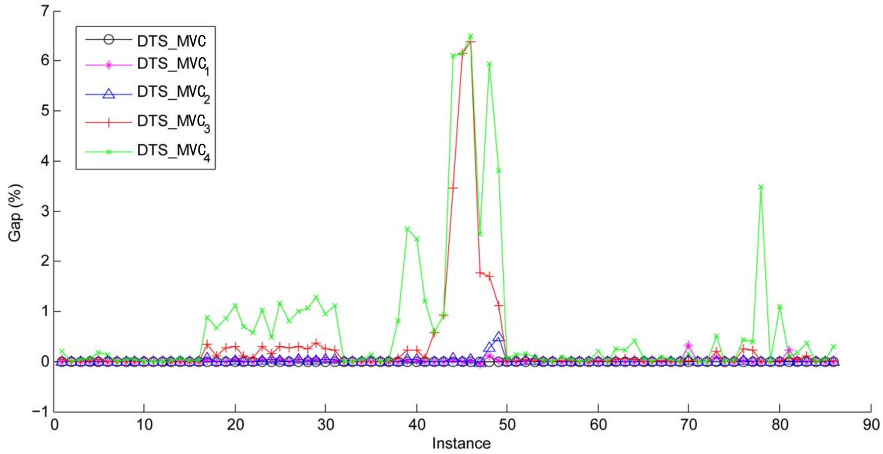

Fig. 2. Gap between DTS_MVC and its algorithm variants in terms of best results.

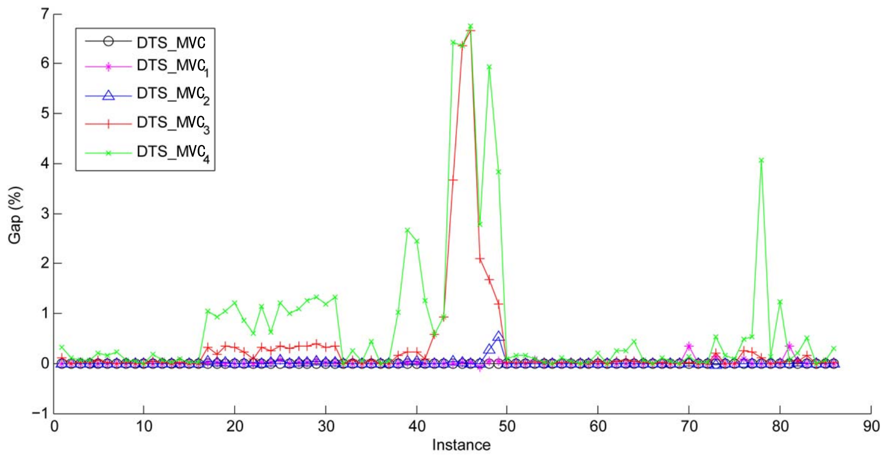

Fig. 3. Gap between DTS_MVC and its algorithm variants in terms of average results.

5 Conclusions and Future Work

We presented a fast and effective stochastic local search algorithm called DTS_MVC for finding small vertex covers in massive sparse graphs. Starting from an initial solution generated by a low-complexity greedy construction procedure, DTS_MVC alternates between a thresholding search phase and a conditional improving phase. Two innovative ingredients of DTS_MVC are the vertex-based strategy and the operation-prohibiting mechanism used in the thresholding search phase. The vertex-based strategy ensures fast solution transformations and enables DTS_MVC to explore various areas of the search space while the operation-prohibiting mechanism combined with a random-vertex-shuffling strategy provides an effective way for DTS_MVC to avoid search cycling. Experimental evaluations on 86 massive real-world graphs showed that DTS_MVC performs very well by competing favorably with two most recent MVC algorithms specially designed for massive graphs. In particular, it is able to discover 7 new best known results (improved upper bounds) never reported in the literature and match the best known results for 64 other cases. The computational results also indicate that DTS_MVC and the existing MVC algorithms complement each other and could be used jointly to handle a large spectrum of graphs. We also conducted additional experiments to understand how each of the main algorithmic components (namely the low-complexity greedy construction procedure, the random-vertex-shuffling strategy, the operation-prohibiting mechanism and the random 'switch' condition) contributes to the performance of DTS_MVC.

DTS_MVC involves a tunable parameter δ which controls the search behavior and the performance of the algorithm. Since fine-tuning this parameter for a given problem instance could lead to improved solutions, it would be interesting to investigate self-adaptive mechanisms to tune this parameter automatically during the search process.

Finally, we note that the proposed vertex-based strategy and the operation-prohibiting mechanism are general enough to be applicable to other graph problems such as those mentioned in the introduction. Since these techniques aim to accelerate the search process and simply the algorithmic procedure, they would particularly be useful to handle large scale problems. It's thus interesting to practically investigate these ideas in the context of solving other search problems in massive sparse graphs.

Acknowledgment

We are grateful to our anonymous reviewers for their insightful comments that have helped us to significantly improve this work. This work was supported by the National Natural Science Foundation of China (Grant No: 71690233), the Region of Pays de la Loire (France, LigeRO project 2009-13) and the Jacques Hadamard Mathematical Foundation (PGMO project 2014-0024H).

References

[1] J. Abello, P. M. Pardalos, M. G. C. Resende, On maximum clique problems in very large graphs. External Memory Algorithms, DIMACS - Series in Discrete Mathematics and Theoretical Computer Science, Volume: 50, American Mathematical Society, 1999, pp. 119–130.

[2] C. Balasubramaniam, S. Butenko, On robust clusters of minimum cardinality in networks. Annals of Operations Research 249(1-2) (2017) 17–37.

[3] V. Boginski, S. Butenko, P. M. Pardalos, Statistical analysis of financial networks. Computational statistics & data analysis 48 (2) (2005) 431–443.

[4] S. Cai, Balance between complexity and quality: local search for minimum vertex cover in massive graphs. Proceedings of Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, July 25-31, 2015, pp. 747–753.

[5] S. Cai, J. Lin, K. Su, Two weighting local search for minimum vertex cover. Proceedings of Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 1107-1113, 2015.

[6] S. Cai, K. Su, Q. Chen, Ewls: A new local search for minimum vertex cover. in: Proceedings of Twenty-Fourth AAAI Conference on Artificial Intelligence, pp. 45-50, 2010.

[7] S. Cai, K. Su, C. Luo, A. Sattar, Numvc: An efficient local search algorithm for minimum vertex cover. Journal of Artificial Intelligence Research (2013) 687–716.

[8] D. Castelino, N. Stephens, Tabu thresholding for the frequency assignment problem. in: Meta-Heuristics, Springer, 1996, pp. 343–359.

[9] Y. Chen, J.K. Hao, Iterated responsive threshold search for the quadratic multiple knapsack problem. Annals of Operations Research 226 (1) (2015) 101–131.

[10] Y. Chen, J.K. Hao, F. Glover, An evolutionary path relinking approach for the quadratic multiple knapsack problem. Knowledge-Based Systems 92 (2016) 23–34.

[11] Y. Chen, J.K. Hao, F. Glover, A hybrid metaheuristic approach for the capacitated arc routing problem. European Journal of Operational Research 253 (1) (2016) 25–39.

[12] G. Dueck, T. Scheuer, Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics 90 (1) (1990) 161–175.

[13] Y. Fan, C. Li, Z. Ma, L. Brankovic, V. Estivill-Castro, A. Sattar, Exploiting reduction rules and data structures: Local search for minimum vertex cover in massive graphs, preprint arXiv:1509.05870, 2015.

[14] S. Fortunato, Community detection in graphs. Physics Reports 486 (3) (2010) 75–174.

[15] J. Gao, J. Chen, M. Yin, R. Chen, Y. Wang, An exact algorithm for Maximum k-Plexes in massive graphs. Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), AAAI, 2018, pp. 1449–1455.

[16] M. R. Garey, D. S. Johnson, Computers and intractability, a guide to the theory of np-completness. San Francisco, Calif.: W.H. Freeman and Co. (1979)

[17] F. Glover, M. Laguna, Tabu search, Springer, 2013.

[18] H. H. Hoos, T. Stützle, Stochastic local search: Foundations & applications, Elsevier, 2004.

[19] Y. Jin, J.K. Hao, General swap-based multiple neighborhood tabu search for the maximum independent set problem. Engineering Applications of Artificial Intelligence 37 (2015) 20–33.

[20] C.-M. Li, Z. Quan, Combining graph structure exploitation and propositional reasoning for the maximum clique problem. Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2010 , Vol. 1, IEEE, 2010, pp. 344–351.

[21] T. Matsunaga, C. Yonemori, E. Tomita, M. Muramatsu, Clique-based data mining for related genes in a biomedical database. BMC Bioinformatics 10 (1) (2009) 205.

[22] M. E. Newman, The structure and function of complex networks. SIAM review 45 (2) (2003) 167–256.

[23] P. R. Östergård, A fast algorithm for the maximum clique problem. Discrete Applied Mathematics 120 (1) (2002) 197–207.

[24] P. M. Pardalos, S. Rebennack, Computational challenges with cliques, quasi-cliques and clique partitions in graphs. in: Experimental Algorithms, Springer, 2010, pp. 13–22.

[25] G. Pastukhov, A. Veremyev, V. Boginski, O. A. Prokopyev, On maximum degreebasedquasiclique problem: Complexity and exact approaches. Networks 71(2) (2018) 136–152.

[26] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W.-k. Liao, A. Choudhary, Fast algorithms for the maximum clique problem on massive sparse graphs. In: Algorithms and Models for the Web Graph, Springer, 2013, pp. 156–169.

[27] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W.-k. Liao, A. Choudhary, Fast algorithms for the maximum clique problem on massive graphs with applications to overlapping community detection. Internet Mathematics 11 (4-5) (2015) 421–448.

[28] B.Q. Pinto, C.C. Ribeiro, I. Rosseti, A. Plastino, A biased randomkey genetic algorithm for the maximum quasi-clique problem. European Journal of Operational Research 271 (3) (2018) 849–865.

[29] W. Pullan, Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. Discrete Optimization 6 (2) (2009) 214–219.

[30] W. Pullan, H. H. Hoos, Dynamic local search for the maximum clique problem. Journal of Artificial Intelligence Research (2006) 159–185.

[31] S. Richter, M. Helmert, C. Gretton, A stochastic local search approach to vertex cover, in: KI 2007: Advances in Artificial Intelligence, Springer, 2007, pp. 412–426.

[32] R. Rossi, N. Ahmed, The network data repository with interactive graph analytics and visualization. Proceedings of Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 4292–4293.

[33] P. San Segundo, A. Lopez, P. M. Pardalos, A new exact maximum clique algorithm for large and massive sparse graphs. Computers & Operations Research 66 (2016) 81–94.

[34] P. San Segundo, D. Rodríguez-Losada, A. Jiménez, An exact bit-parallel algorithm for the maximum clique problem. Computers & Operations Research 38 (2) (2011) 571–581.

[35] S. Shahinpour, S. Shirvani, Z. Ertem, S. Butenko, Scale reduction techniques for computing Maximum Induced Bicliques. Algorithms 10(4) (2017) 113.

[36] S. J. Shyu, P.-Y. Yin, B. M. Lin, An ant colony optimization algorithm for the minimum weight vertex cover problem. Annals of Operations Research 131 (1-4) (2004) 283–304.

[37] C. D. Tarantilis, C. T. Kiranoudis, V. S. Vassiliadis, A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. European Journal of Operational Research 152 (1) (2004) 148–158.

[38] E. Tomita, T. Kameda, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. Journal of Global Optimization 37 (1) (2007) 95–111.

[39] A. Verma, A. Buchanan, S. Butenko, Solving the maximum clique and vertex coloring problems on very large sparse networks. INFORMS Journal on Computing 27 (1) (2015) 164–177.

[40] Y. Wang, S. Cai, M. Yin, New heuristic approaches for maximum balanced biclique problem. Information Sciences 432 (2018) 362–375.

[41] Q. Wu, J.K. Hao, An adaptive multistart tabu search approach to solve the maximum clique problem. Journal of Combinatorial Optimization 26 (1) (2013) 86–108.

[42] Q. Wu, J.K. Hao, A review on algorithms for maximum clique problems. European Journal of Operational Research 242 (3) (2015) 693–709.

[43] Q. Wu, J.K. Hao, Solving the winner determination problem via a weighted maximum clique heuristic. Expert Systems with Applications 42 (1) (2015) 355–365.

[44] O. Yezerska, S. Butenko, V. L. Boginski, Detecting robust cliques in graphs subject to uncertain edge failures. Annals of Operations Research 262(1) (2018) 109–132.

[45] Y. Zhou, J.K. Hao, Frequency-driven tabu search for the maximum s-plex problem. Computers & Operations Research 86 (2017) 65–78.

[46] Y. Zhou, J.K. Hao, Tabu search with graph reduction for finding maximum balanced bicliques in bipartite graphs. Engineering Applications of Artificial Intelligence 77 (2018) 86–97.

A Appendix

This appendix shows the detailed results of our DTS_MVC algorithm in comparison with the state-of-the-art FastVC and LinCom algorithms on the 86 well-known massive graphs.

Table A.1
Comparative results of DTS_MVC with FastVC [4] and LinCom [13] on biological networks, collaboration networks, facebook networks, interaction networks, infrastructure networks, recommend networks and scientific networks (totally 49 instances). The best of the f_{best} values are in boldface and the best of the f_{avg} values are in italic. The best result of DTS_MVC is starred if it improves on the current best known result BKR. n/a denotes "not available".

Instance			BKR	DTS_MVC				FastVC			LinCom	
Graph	$\lvert V\rvert$	$\lvert E\rvert$		δ	f_{best}	f_{avg}	CPU(s)	f_{best}	f_{avg}	CPU(s)	f_{best}	f_{avg}
bio-dmela	7393	25569	2630	1	**2630**	*2630.02*	1.47	2632	2632.00	0.03	n/a	n/a
bio-yeast	1458	1948	456	1	**456**	*456.00*	0.00	**456**	*456.00*	0.00	n/a	n/a
ca-AstroPh	17903	196972	11483	1	**11483**	*11483.00*	0.00	**11483**	*11483.00*	0.06	**11483**	11483.01
ca-citeseer	227320	814134	129193	1	**129193**	*129193.00*	5.15	**129193**	*129193.00*	2.59	**129193**	129193.36
ca-coauthors-dblp	540486	15245729	472179	1	**472179**	*472179.00*	369.50	**472179**	472179.13	54.42	**472179**	472179.02
ca-CondMat	21363	91286	12480	1	**12480**	*12480.00*	0.00	**12480**	*12480.00*	0.10	**12480**	12480.06
ca-CSphd	1882	1740	550	1	**550**	*550.00*	0.00	**550**	*550.00*	0.00	n/a	n/a
ca-dblp-2010	226413	716460	121969	1	**121969**	*121969.00*	7.09	**121969**	*121969.00*	13.18	**121969**	121969.64
ca-dblp-2012	317080	1049866	164949	1	**164949**	*164949.00*	7.24	**164949**	*164949.00*	6.63	**164949**	164950.35
ca-Erdos992	6100	7515	461	1	**461**	*461.00*	0.00	**461**	*461.00*	0.00	n/a	n/a
ca-GrQc	4158	13422	2208	1	**2208**	*2208.00*	0.00	**2208**	*2208.00*	0.01	n/a	n/a
ca-HepPh	11204	117619	6555	1	**6555**	*6555.00*	0.00	**6555**	*6555.00*	0.03	n/a	n/a
ca-hollywood-2009	1069126	56306653	864052	1	**864052**	*864052.00*	328.40	**864052**	*864052.00*	75.76	**864052**	864052.01
ca-MathSciNet	332689	820644	139951	1	**139951**	*139951.00*	12.96	**139951**	*139951.00*	9.09	**139951**	139952.23
socfb-A-anon	3097165	23667394	375230	1	**375230**	375233.00	63.46	375231	375233.46	259.74	**375230**	*375230.82*
socfb-B-anon	2937612	20959854	303048	1	**303048**	303049.38	34.36	**303048**	303049.00	220.20	**303048**	*303048.00*
socfb-Berkeley13	22900	852419	17210	1	**17210**	*17213.20*	366.53	**17210**	17214.13	315.58	**17210**	17215.93
socfb-CMU	6621	249959	4986	1	**4986**	*4986.76*	12.82	**4986**	4986.86	10.57	**4986**	4987.24
socfb-Duke14	9885	506437	7683	1	**7683**	*7683.39*	164.56	**7683**	7683.89	243.35	**7683**	7684.98
socfb-Indiana	29732	1305757	23315	1	**23315**	*23317.60*	430.06	**23315**	23319.69	462.64	23319	23323.79
socfb-MIT	6402	251230	4657	1	**4657**	*4657.00*	162.38	**4657**	4657.01	153.90	**4657**	4657.56
socfb-OR	63392	816886	36548	1	**36548**	36550.00	464.07	**36548**	36550.25	276.70	**36548**	*36549.50*
socfb-Penn94	41536	1362220	31162	1	**31160***	*31166.00*	545.29	31162	31167.56	592.00	31165	31170.78
socfb-Stanford3	11586	568309	8517	1	**8517**	*8517.97*	23.08	8518	8518.00	27.78	8518	8518.35
socfb-Texas84	36364	1590651	28167	1	**28167**	*28172.30*	497.12	**28167**	28174.53	587.62	28169	28178.98
socfb-UCLA	20453	747604	15223	1	**15222***	*15224.90*	340.07	15223	15225.61	366.27	15224	15228.85
socfb-UConn	17206	604867	13230	2	**13230**	*13232.10*	333.28	**13230**	13232.41	317.04	13232	13235.99
socfb-UCSB37	14917	482215	11261	1	**11261**	*11263.20*	258.23	**11261**	11264.14	305.43	11262	11265.54
socfb-UF	35111	1465654	27306	1	**27305***	*27309.80*	477.22	27306	27311.76	543.21	27310	27316.25
socfb-UIllinois	30795	1264421	24091	3	**24091**	*24095.80*	507.17	**24091**	24096.31	533.38	24095	24101.18
socfb-Wisconsin87	23831	835946	18383	1	**18383**	*18386.30*	370.48	**18383**	18387.59	471.40	18384	18390.13
ia-email-EU	32430	54397	820	1	**820**	*820.00*	0.00	**820**	*820.00*	0.00	n/a	n/a
ia-email-univ	1133	5451	594	1	**594**	*594.00*	0.00	**594**	*594.00*	0.00	n/a	n/a
ia-enron-large	33696	180811	12781	1	**12781**	*12781.00*	0.00	**12781**	*12781.00*	0.11	**12781**	12781.20
ia-fb-messages	1266	6451	578	1	**578**	*578.00*	0.00	**578**	*578.00*	0.01	n/a	n/a
ia-reality	6809	7680	81	1	**81**	*81.00*	0.00	**81**	*81.00*	0.00	n/a	n/a
ia-wiki-Talk	92117	360767	17288	1	**17288**	*17288.00*	0.04	**17288**	*17288.00*	0.22	n/a	n/a
inf-power	4941	6594	2203	1	**2203**	*2203.00*	0.00	**2203**	*2203.00*	0.03	**2203**	2203.01
inf-roadNet-CA	1957027	2760388	1001058	1	1001158	1001253.63	969.00	1001273	1001320.00	974.57	**1001058**	*1001139.61*
inf-roadNet-PA	1087562	1541514	555035	1	555160	555231.23	921.42	555220	555242.00	850.93	**555035**	*555107.22*
rec-amazon	91813	125704	47605	1	**47605**	*47605.55*	225.65	47607	47607.40	1.79	**47605**	47605.62
sc-ldoor	952203	20770807	856755	8	856759	856762.87	788.41	**856755**	*856757.00*	313.25	**856755**	856757.18
sc-msdoor	415863	9378650	381558	1	**381558**	381559.72	594.24	**381558**	*381559.00*	54.00	381559	381559.86
sc-nasasrb	54870	1311227	51243	0	**51242***	*51246.96*	691.44	51244	51247.20	506.26	51243	51249.23
sc-pkustk11	87804	2565054	83911	0	**83911**	83913.93	125.97	**83911**	83913.70	37.73	**83911**	*83913.52*
sc-pkustk13	94893	3260967	89217	0	89220	89224.61	773.88	**89218**	*89221.20*	577.88	89219	89222.95
sc-pwtk	217891	5653221	207698	1	**207695***	*207707.18*	896.61	207716	207709.00	443.51	207698	207711.11
sc-shipsec1	140385	1707759	117278	1	**117231***	*117295.35*	936.47	117348	117381.00	842.21	117278	117319.88
sc-shipsec5	179104	2200076	146991	1	147081	147128.31	898.25	147137	147170.00	574.72	**146991**	*147022.95*

Table A.2
Comparative results of DTS_MVC with FastVC [4] and LinCom [13] on the social networks, technological networks and web link networks (totally 37 instances). The best of the f_{best} values are in boldface and the best of the f_{avg} values are in italic. The best result of DTS_MVC is starred if it improves on the current best known result BKR. n/a denotes "not available".

Instance			BKR	DTS_MVC				FastVC			LinCom	
Graph	$\|V\|$	$\|E\|$		δ	f_{best}	f_{avg}	CPU(s)	f_{best}	f_{avg}	CPU(s)	f_{best}	f_{avg}
soc-BlogCatalog	88784	2093195	20752	1	**20752**	*20752.00*	16.59	**20752**	20752.01	0.61	**20752**	20752.01
soc-brightkite	56739	212945	21190	1	**21190**	*21190.01*	51.37	**21190**	21190.10	0.40	**21190**	21190.09
soc-buzznet	101163	2763066	30613	4	30617	30625.79	548.02	30625	30625.50	239.30	**30613**	*30613.00*
soc-delicious	536108	1365961	85319	4	85824	85933.04	937.86	86184	86224.40	10.58	**85319**	*85333.75*
soc-digg	770799	5907132	103234	1	103242	103244.69	91.34	103244	103245.00	4.73	**103234**	*103234.01*
soc-douban	154908	327162	8685	1	**8685**	*8685.00*	0.00	**8685**	*8685.00*	0.01	n/a	n/a
soc-epinions	26588	100120	9757	1	**9757**	*9757.00*	0.96	**9757**	9757.85	0.28	**9757**	9757.02
soc-flickr	513969	3190452	153271	1	**153271**	153271.52	192.07	153272	153272.00	63.51	**153271**	*153271.45*
soc-flixster	2523386	7918801	96317	1	**96317**	*96317.00*	23.25	**96317**	*96317.00*	2.88	**96317**	96317.02
soc-FourSquare	639014	3214986	90108	1	**90108**	90108.99	88.71	**90108**	90109.30	71.72	**90108**	*90108.13*
soc-gowalla	196591	950327	84222	1	**84222**	84222.54	445.45	**84222**	84222.40	46.37	**84222**	*84222.07*
soc-lastfm	1191805	4519330	78688	1	**78688**	78688.01	5.35	**78688**	*78688.00*	2.27	n/a	n/a
soc-livejournal	4033137	27933062	1868924	1	1869051	1869074.41	976.36	1869060	1869082.00	973.69	**1868924**	*1868932.92*
soc-LiveMocha	104103	2193083	43427	1	**43427**	43427.53	377.29	**43427**	*43427.00*	176.74	n/a	n/a
soc-pokec	1632803	22301964	843344	1	843415	843431.68	941.55	843422	843438.00	868.47	**843344**	*843347.38*
soc-slashdot	70068	358647	22373	1	**22373**	*22373.00*	0.51	**22373**	*22373.00*	0.49	n/a	n/a
soc-twitter-follows	404719	713319	2323	1	**2323**	*2323.00*	0.00	**2323**	*2323.00*	0.07	n/a	n/a
soc-youtube	495957	1936748	146376	1	**146376**	*146376.00*	67.91	**146376**	146376.34	13.02	**146376**	146376.10
soc-youtube-snap	1134890	2987624	276945	1	**276945**	*276945.00*	80.49	276945	*276945.00*	27.71	**276945**	276945.21
tech-as-caida2007	26475	53381	3683	1	**3683**	*3683.00*	14.25	3684	3684.00	0.01	n/a	n/a
tech-as-skitter	1694616	11094209	525086	1	527269	527402.17	945.33	527587	527636.00	696.36	**525086**	*525099.14*
tech-internet-as	40164	85123	5700	1	**5700**	*5700.00*	0.00	**5700**	*5700.00*	0.05	n/a	n/a
tech-p2p-gnutella	62561	147878	15682	1	**15682**	*15682.00*	4.95	**15682**	*15682.00*	0.05	n/a	n/a
tech-RL-caida	190914	607610	74607	10	74719	74757.60	980.67	74930	74942.50	10.75	**74607**	*74615.25*
tech-routers-rf	2113	6632	795	1	**795**	*795.00*	0.00	**795**	*795.00*	0.00	n/a	n/a
tech-WHOIS	7476	56943	2284	1	**2284**	*2284.00*	0.00	**2284**	*2284.00*	0.01	n/a	n/a
web-arabic-2005	163598	1747269	114420	4	114426	114433.78	564.57	114429	114432.00	473.07	**114420**	*114420.67*
web-BerkStan	12305	19500	5384	1	**5384**	*5384.00*	129.30	**5384**	5384.72	6.82	**5384**	5384.13
web-edu	3031	6474	1451	1	**1451**	*1451.00*	0.00	**1451**	*1451.00*	0.00	n/a	n/a
web-google	1299	2773	498	1	**498**	*498.00*	0.00	**498**	*498.00*	0.00	n/a	n/a
web-indochina-2004	11358	47606	7300	1	**7300**	*7300.00*	0.89	**7300**	*7300.00*	0.31	n/a	n/a
web-it-2004	509338	7178413	414646	1	**414616***	414651.86	422.10	414671	414675.00	22.78	414646	*414649.00*
web-sk-2005	121422	334419	58173	1	**58173**	58178.39	180.16	**58173**	*58173.00*	22.26	n/a	n/a
web-spam	4767	37375	2297	1	**2297**	*2297.00*	0.15	2298	2298.00	0.03	**2297**	2297.26
web-uk-2005	129632	11744049	127774	1	**127774**	*127774.00*	0.00	**127774**	*127774.00*	0.08	n/a	n/a
web-webbase-2001	16062	25593	2651	1	**2652**	*2652.00*	0.00	**2652**	*2652.00*	0.04	n/a	n/a
web-wikipedia2009	1864433	4507315	648300	1	648312	648324.46	816.33	648317	648320.00	808.05	**648300**	*648312.39*