IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 20, 2018, accepted September 30, 2018, date of publication October 15, 2018,
date of current version November 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2875499

An Efficient Local Search Algorithm for the Minimum k-Dominating Set Problem

RUIZHI LI[1], HUAN LIU[2], XIAOLI WU[2], JUN WU[2], AND MINGHAO YIN[1,2,3]
[1]Jilin University of Finance and Economics, Changchun 130117, China
[2]Northeast Normal University, Changchun 130117, China
[3]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun 130024, China

Corresponding author: Minghao Yin (ymh@nenu.edu.cn)

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant 61763003, Grant 61502111, Grant 61806082, and Grant 61503074.

ABSTRACT The minimum k-dominating set (MKDS) problem, a generalization of the classical minimum dominating set problem, is an important NP-hard combinatorial optimization problem with various applications. First, to alleviate the cycling problem in the local search, a MKDS two-level configuration checking (MKDSCC^2) strategy is presented. Second, we use the vertex cost scheme to define the scoring mechanism and to improve the solution effectively. Third, by combining MKDSCC^2 strategy and the scoring mechanism, we propose a vertex selection strategy to decide which vertex should be added into or removed from the candidate solution. Based on these strategies, an efficient local search algorithm (VSCC^2), which incorporates a two-level configuration checking strategy, scoring mechanism, and vertex selection strategy, is proposed. We compare the performance of VSCC^2 with the classic GRASP algorithm and the famous commercial solver CPLEX on the classical instances. The comprehensive results show that the VSCC^2 algorithm is very competitive in terms of solution quality and computing time.

INDEX TERMS Heuristic, local search, minimum k-dominating set problem.

I. INTRODUCTION

Given an undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set, a dominating set (DS) of G is a subset $D \subseteq V$ such that every vertex in $V \backslash D$ is adjacent to at least one vertex in D. The minimum dominating set (MDS) problem aims to find the dominating set with the minimum size. A k-dominating set (KDS) of G is a subset $D_k \subseteq V$ such that every vertex in $V \backslash D_k$ is adjacent to at least k vertices in D_k [1]. The minimum k-dominating set problem aims to find the k-dominating set with the minimum size. The MKDS problem can be viewed as a generalized version of the MDS problem. Specifically, the minimum 1-dominating set problem is equivalent to the minimum dominating set problem.

The minimum dominating set problem has various applications in real-word domains such as routing in wireless networks [2]–[4], document processing [5], [6], and social networks [7]. Whereas sometimes the dominating set problem cannot better model the actual application problems. For example, in a wireless ad-hoc network, the level of service required by a dominatee cannot be accomplished by only one dominator, and they call for collect services from several dominators to meet the dominatee' needs. Even if any k -1 dominator fails, each dominatee is guaranteed to connect to at least one dominator [8], [9]. The problem can be modelled as minimum k-dominating set problem. Therefore, MKDS problem has a stronger modeling capability and has wider applications in several diverse areas [10]–[12].

In the past two decades, various algorithms have been proposed to solve the MDS problem, and they can be mainly classified into exact and heuristic algorithms. Exact algorithms [13]–[16] are mostly based on branch-bound method or branch-cut algorithm. Exact algorithms have the advantage of ensuring the optimal solutions, but they require a computing time, in general, exponential growth with the size of the problem. So various heuristic algorithms have been devised to handle the minimum dominating set problem. Hedar and Ismail [17] proposed an algorithm HGA-MDS, which is based on genetic algorithm (GA) to handle the MDS problem. After that Giap and Ha [18] designed a good parallel genetic algorithm (PGAs) model for MDS problem. Chalupa [19] presented an order-based randomized local search (RLSo) algorithm to compute MDS problem indirectly by employing a representation based on permutations of vertices, which are transformed into dominating sets using a greedy algorithm. Hedar and Ismail [20] again proposed a method SAMDS based on simulated annealing (SA) to solve the minimum dominating set problem. Compared with the minimum dominating set problem, there are relatively few algorithms to solve the minimum k-dominating set problem.

In this paper, a new local search framework is proposed for the minimum k-dominating set problem based on some new ideas. Firstly, during the local search process, most local search algorithms are subject to the cycling problem. To tackle this problem, two-level configuration checking strategy (CC^2) is recently proposed in [21]. The CC^2 strategy has been successfully used in the minimum weight dominating set problem. In this strategy, the configuration of a vertex v is the state of the neighborhood $N(v)$ and the neighborhood of each vertex in $N(v)$. For a vertex, if its configuration has not been changed after the last time it was removed from the candidate solution, it will be banned from being added into the candidate solution. On this basis, we adapt the two-level configuration checking strategy into the minimum k-dominating set problem during the local search process.

Secondly, the scoring strategy has a significant role during the local search [22]–[28]. In our work, we propose a new scoring strategy based the vertex cost to obtain more promising solutions and increase the diversity of solutions. For the scoring strategy, the score value of each vertex can be modified dynamically.

Furthermore, by integrating the scoring strategy with two-level configuration checking strategy, we design a vertex selection strategy to decide which vertex should be added into or removed from the candidate solution.

Finally, by merging all the above strategies, we develop a local search algorithm named $VSCC^2$ to solve the MKDS problem. To measure the efficiency of $VSCC^2$, the experimental results of our algorithm are compared with those of commercial solver CPLEX and the classic GRASP algorithm, and our algorithm obtains better solutions than or same solutions as CPLEX and GRASP on almost all instances.

The remainder of this paper is constructed as follows. In Section 2, some useful notations are introduced. The two-level configuration checking strategy for the minimum k-dominating set problem is proposed in Section 3. Furthermore, the scoring strategy is introduced in Section 4. And then a new vertex selection strategy is proposed in Section 5. Then, a detailed description of $VSCC^2$ is described in Section 6. In Section 7, the experimental results will be listed. Finally, conclusions and future directions are shown in the last section.

II. PRELIMINARY

At first, we shall introduce some background information for MKDS problem. An undirected graph $G = (V, E)$ consists of a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E = \{e_1, e_2, \ldots, e_m\}$, where each edge $e = (v, u)$ connects two vertices u and v, and we say that vertices u and v are the endpoints of edge e. We shall use $dist\ (u, v)$, which is the number of edges in a shortest path from u to v, to denote the distance between two vertices u and v. Given a candidate solution D_k, $s_i \in \{1, 0\}$ denotes the state of vertex v_i, where $s_i = 1$ means $v_i \in D_k$, and $s_i = 0$ means $v_i \notin D_k$. We shall use $m(D_k)$ to denote the number of vertices in D_k. For a vertex v, we shall use $N_i(v) = \{u|dist(u, v) = i\}$ to denote the ith level neighborhood of the vertex v, and we denote $N^k(v) = \cup_{i=1}^{k} N_i(v)$. And the first-level neighborhood $N_1(v)$ is the same as $N(v)$, and we shall use $N[v] = N(v) \cup \{v\}$ to denote the closed neighbor set of v.

TABLE 1. Different values of k.

k	Value
k_{min}	2
k_{max}	$\lceil max\{degree(v), for\ v \in V\}/2\rceil$
k_{med}	$\lceil (k_{min} + k_{max})/2\rceil$

TABLE 2. Different values of p.

Instance	k	p
General graphs	k_{min}	0.15
General graphs	k_{med}	0.15
General graphs	k_{max}	0.85
UDG	k_{min}	0.15
UDG	k_{med}	0.15
UDG	k_{max}	0.85
DIMACS	k_{min}	0.15
DIMACS	k_{med}	0.75
DIMACS	k_{max}	0.95

Definition 1 (Dominating Set, DS): Given an undirected graph $G\ (V, E)$, the dominating set of G is a vertex subset $D \subseteq V$ such that every vertex in $V \backslash D$ has at least one neighbor in D.

Definition 2 (Minimum Dominating Set, MDS): Given an undirected graph $G(V, E)$, the minimum dominating set problem calls for finding a dominating set D with minimum cardinality.

Definition 3 (k-Dominating Set, KDS): Given an undirected graph $G\ (V, E)$, the k-dominating set of G is a vertex subset $D_k \subseteq V$ such that every vertex in $V \backslash D_k$ is adjacent to at least k vertices in D_k.

Definition 4 (Minimum k-Dominating Set, MKDS): Given an undirected graph $G(V, E)$, the minimum k-dominating set problem calls for finding a k-dominating set D_k with minimum cardinality.

The definitions show that dominating set problem can be viewed as a special problem of the k-dominating set when k equals 1. During the local search process, our algorithm maintains a candidate solution $D_k \subseteq V$. For a candidate solution D_k, vertices that belong to D_k are called dominating vertices. If a vertex v which is adjacent to at least k vertices in D_k, is called k-dominated vertex, otherwise it's called non-k-dominated vertex. For a vertex, its age denotes the number of steps when it is selected.

III. TWO-LEVEL CONFIGURATION CHECKING STRATEGY

Local search algorithms often visit a candidate solution repeatedly during the search process. This phenomenon is

TABLE 3. The comparative results of CPLEX, GRASP, and VSCC^2 with k_{min} on the general graphs benchmarks.

Instance	k_{min}	CPLEX	GRASP			VSCC^2		
			Min	Avg	Avgtime	Min	Avg	Avgtime
50_50	2	**26**	**26**	**26**	8.54	**26**	**26**	**0.01**
50_100	2	**21**	**21**	**21**	51.2	**21**	**21**	**0**
50_250	2	**10**	**10**	**10**	0.11	**10**	**10**	**0**
50_500	2	**6**	**6**	**6**	0.83	**6**	**6**	**0**
50_750	2	**4**	**4**	**4**	0.08	**4**	**4**	**0**
50_1000	2	**3**	**3**	**3**	0.01	**3**	**3**	**0**
100_100	2	**51**	52	52	56.17	**51**	**51**	**0.01**
100_250	2	**34**	36	36.2	220.71	**34**	**34**	**0.03**
100_500	2	**20**	21	21	1.64	**20**	**20**	**0.01**
100_750	2	**15**	**15**	**15**	2.53	**15**	**15**	**0**
100_1000	2	**12**	**12**	**12**	32.53	**12**	**12**	**0**
100_2000	2	**7**	**7**	**7**	2.93	**7**	**7**	**0**
150_150	2	**76**	87	87.8	466	**76**	**76**	**0.02**
150_250	2	**65**	70	70.9	304.54	**65**	**65**	**0.13**
150_500	2	**42**	45	46.1	194.9	**42**	**42**	**0.04**
150_750	2	**31**	34	34.8	244.91	**31**	**31**	**0.68**
150_1000	2	**25**	27	27.9	243.08	**25**	**25**	**0.37**
150_2000	2	**15**	16	16.4	200.6	**15**	**15**	**0.01**
150_3000	2	**10**	11	11.3	211.89	**10**	**10**	**0.08**
200_250	2	**96**	105	107.3	334.98	**96**	**96**	**0.07**
200_500	2	**67**	78	79	518.71	**67**	**67**	**18.39**
200_750	2	**52**	60	61.9	339.36	**52**	**52**	**0.05**
200_1000	2	**<=42**	47	47.8	163.34	**42**	**42**	**0.03**
200_2000	2	<=25	28	28	145.69	**24**	**24**	**0.04**
200_3000	2	<=18	20	20.5	251.95	**17**	**17**	**0.91**
250_250	2	**126**	151	151.2	481.1	**126**	**126**	**0.03**
250_500	2	**97**	112	113.5	291.48	**97**	**97**	**5.65**
250_750	2	**75**	89	89.8	411.82	**75**	**75.3**	**62.09**
250_1000	2	<=63	72	73.5	352.32	**61**	**61**	**9.63**
250_2000	2	<=37	43	43.3	317.09	**36**	**36**	**4.94**
250_3000	2	<=28	30	30.9	461.79	**26**	**26**	**0.8**
250_5000	2	**<=18**	20	21	249.41	**18**	**18**	**0.03**
300_300	2	**151**	182	183	347.69	**151**	**151**	**0.04**
300_500	2	**127**	148	148.7	403.09	**127**	**127.7**	**188.75**
300_750	2	**101**	121	121.4	175.64	**101**	**101.7**	**5.3**
300_1000	2	**<=83**	98	98.6	480.04	**83**	**83**	**5.25**
300_2000	2	<=51	57	57.4	554.22	**49**	**49**	**0.36**
300_3000	2	<=38	43	43.9	323.19	**36**	**36**	**29.11**
300_5000	2	<=25	28	29.8	225.58	**24**	**24**	**20.92**
500_500	2	**251**	310	311.4	568.01	**251**	**251**	**0.14**
500_1000	2	**<=195**	230	231.8	471.26	196	196.1	**150.09**
500_2000	2	<=123	147	149.1	344.71	**121**	**121.1**	**58.67**

TABLE 3. *(Continued.)* **The comparative results of CPLEX, GRASP, and VSCC^2 with k_{min} on the general graphs benchmark.**

500_5000	2	<=63	75	76.3	465.81	**61**	**61**	**2.18**
500_10000	2	<=37	44	44.8	665.66	**36**	**36**	**1.55**
800_800	2	**401**	505	506.6	392.48	**401**	**401**	**0.43**
800_1000	2	**384**	451	452.4	406.2	386	386.1	**39.36**
800_2000	2	**<=271**	326	329.5	433.18	272	273	**280.92**
800_5000	2	<=147	173	173.9	275.61	**141**	**141.2**	**154.53**
800_10000	2	<=87	104	105.4	433.14	**82**	**82.8**	**31.34**
1000_1000	2	**501**	631	635.3	400.33	**501**	**501**	**0.85**
1000_5000	2	<=216	252	254.8	344.21	**209**	**210**	**147.28**
1000_10000	2	<=132	155	156.7	453.83	**124**	**124.8**	**151.94**
1000_15000	2	<=96	114	115.4	358	**90**	**90.1**	**151.81**
1000_20000	2	<=79	92	92.9	356.79	**72**	**72.7**	**89.41**

called the cycling problem, which not only wastes time, but also makes the algorithm often fall into the local optimum and reduces the performance of the algorithm. To alleviate the cycling problem, the two-level configuration checking (CC^2) strategy was proposed to handle this problem in local search. The CC^2 strategy has been successfully used to solve the minimum weight dominating set problem. Therefore, we adapt this strategy to the minimum k-dominating set problem during the local search process.

Furthermore, we shall introduce the definition of two-level configuration checking in minimum k-dominating set problem, and we call it minimum k-dominating set two-level configuration checking (MKDSCC^2). For the MKDSCC^2 strategy, the configuration of a vertex is represented as a vector consisting of the states of all vertices in $N^2(v)$. For a vertex $v \notin D_k$, if at least one vertex in $N^2(v)$ has changed its state since the last selection, then the configuration of v is changed.

To implement MKDSCC^2 strategy, we introduce a Boolean array MKDSCC^2 whose size equals the number of vertices in the graph. For a vertex v, the value of MKDSCC^2 means whether its configuration has changed since the recent state change of v. If MKDSCC$^2[v] = 1$, it means that v is a configuration changed vertex and could be picked in the next adding procedure, otherwise MKDSCC$^2[v] = 0$. Based on this, the MKDSCC$^2[v]$ array is maintained as follows.

MKDSCC^2-RULE1. In the initial process, for each vertex v, MKDSCC$^2[v]$ is initialized as 1.

MKDSCC^2-RULE2. When a vertex v is removed from D_k, MKDSCC$^2[v]$ is reset to 0 immediately. Then for each vertex $u \in N^2(v)$, MKDSCC$^2[u]$ is set to 1.

MKDSCC^2-RULE3. When a vertex v is added to D_k, for each vertex $u \in N^2[v]$, MKDSCC$^2[u]$ is set to 1.

IV. SCORING STRATEGY

In the local search process, deciding which vertex should be added into or removed from the candidate solution D_k plays an important role during the process of local search. Fortunately, the MKDSCC^2 strategy can contribute to alleviate the cycling problem. We shall introduce a scoring mechanism for the MKDS problem. And we use this strategy together with MKDSCC^2 strategy, random walk [29], [30] and tabu strategy [31], [32] to decide which vertex should be picked as a solution component.

In a graph, each vertex v is associated with a vertex *cost* property, denoted by $cost(v)$. In the initialization process, the *cost* of each vertex is set to be 1. After each iteration of local search, each vertex v will be checked whether v is k-dominated by the candidate solution D_k. If v is non-k-dominated, $cost(v)$ is increased by one. Based on this, we propose a new score function for the minimum k-dominating set problem. It is defined as below.

Definition 5 Given an undirected graph $G(V, E)$, and a candidate solution D_k, the cost based scoring function denoted by *score*, is denoted in formula (1) and formula (2).

for $u \notin D_k$:

$$score(u) = \sum_{v \in M_1 \wedge Z_v < k} cost\,(v) \tag{1}$$

for $u \in D_k$:

$$score\,(u) = \begin{cases} -\sum_{v \in M_2 \wedge Z_v = k} cost\,(v)\,, & if\ Z_u > k \\ -\sum_{v \in M_2 \wedge Z_v = k} cost\,(v) - cost\,(u)\,, & otherwise \end{cases} \tag{2}$$

Where $M_1 = N[u] \setminus D_k$, $M_2 = N(u) \setminus D_k$, Z_v represents the number of neighbors of the vertex v in the candidate solution, and Z_u represents the number of neighbors of the vertex u in the candidate solution. When a vertex $u \notin D_k$, the score of u is the sum of the costs of vertices which are not in D_k and non-k-dominated in the closed neighbor set of u. When a vertex $u \in D_k$, we shall consider two situations. If $Z_u > k$, the score of u is the opposite number of the sum of the costs of vertices (not in the D_k) whose Z_v are equal to k in the neighbor set of u. Otherwise the score of u is the opposite number of the

TABLE 4. The comparative results of CPLEX, GRASP, and VSCC^2 with k_{med} on the general graphs benchmarks.

			GRASP			VSCC^2		
Instance	k_{med}	CPLEX	Min	Avg	Avgtime	Min	Avg	Avgtime
50_50	2	**26**	**26**	**26**	6.27	**26**	**26**	**0**
50_100	3	**28**	**28**	**28**	105.16	**28**	**28**	**0**
50_250	6	**25**	26	26.9	29.35	**25**	**25**	**0.01**
50_500	8	**19**	**19**	19.7	181.5	**19**	**19**	**0**
50_750	10	**16**	17	17	0.17	**16**	**16**	**0.01**
50_1000	12	**15**	**15**	**15**	0.1	**15**	**15**	**0**
100_100	2	**51**	52	52	75.19	**51**	**51**	**0.01**
100_250	4	**59**	62	62.2	348.77	**59**	**59**	**0.03**
100_500	5	**44**	46	46	237.22	**44**	**44**	**0.02**
100_750	7	**43**	46	46.2	280.07	**43**	**43.4**	**297.16**
100_1000	9	<=43	45	45.4	204.6	**42**	**42**	**0.04**
100_2000	14	<=**35**	37	37.6	195.23	**35**	**35**	**0.02**
150_150	2	**76**	87	87.8	479.35	**76**	**76**	**0.01**
150_250	3	**90**	92	92.8	312.79	**90**	**90**	**0.02**
150_500	4	<=74	80	80.6	193.32	**73**	**73**	**5.87**
150_750	5	<=**65**	72	72.6	197.51	**65**	**65**	**0.3**
150_1000	7	<=**70**	78	78.9	168.44	**70**	**70**	**91.4**
150_2000	11	<=61	69	70	462.03	**60**	**60.1**	**357.59**
150_3000	15	<=57	62	62.9	233.5	**55**	**55.3**	**313.03**
200_250	2	**96**	105	107.1	422.13	**96**	**96**	**0.12**
200_500	4	**116**	126	127.2	321.21	**116**	**116**	**0.12**
200_750	5	<=**109**	122	122.7	525.27	**109**	**109**	**0.79**
200_1000	6	<=105	118	119.5	317.59	**103**	**103.8**	**417.36**
200_2000	9	<=87	96	96.9	586.67	**85**	**85.1**	**247.07**
200_3000	12	<=80	88	88.8	287.33	**77**	**77**	**302.77**
250_250	2	**126**	151	151.2	394.16	**126**	**126**	**0.04**
250_500	3	**134**	149	149.7	517.59	**134**	**134.1**	**2.59**
250_750	4	<=132	148	149.5	443.33	**131**	**131**	**172.98**
250_1000	5	<=132	150	150.9	485.77	**131**	**131.2**	**76.64**
250_2000	8	<=117	136	136.9	232.34	**116**	**116.7**	**161.53**
250_3000	10	<=103	114	115.3	368.78	**100**	**100.7**	**174.33**
250_5000	16	<=103	113	113.8	385.04	**98**	**98**	**158.12**
300_300	2	**151**	182	183	293.66	**151**	**151**	**0.05**
300_500	3	**182**	193	193.9	422.12	**182**	**182**	**0.18**
300_750	4	<=**178**	196	197.3	337.37	**178**	**178**	**4.06**
300_1000	4	<=148	168	169.7	339.74	**146**	**146.1**	**75.63**
300_2000	8	<=**161**	187	187.4	371.82	**161**	**162.1**	**283.01**
300_3000	10	<=146	163	165.3	415.78	**139**	**139.9**	**242.19**
300_5000	14	<=129	140	140.6	307.84	**123**	**123**	**58.79**
500_500	2	**251**	310	311.8	466.39	**251**	**251**	**0.14**
500_1000	3	<=270	308	309.9	188.28	**269**	**269.3**	**71.17**

TABLE 4. ***(Continued.)*** **The comparative results of CPLEX, GRASP, and VSCC^2 with k_{med} on the general graphs benchmarks.**

500_2000	5	<=264	304	307.8	481.44	**259**	**260.2**	**215.88**
500_5000	9	<=221	256	258.2	515.71	**214**	**214.3**	**177.8**
500_10000	15	<=200	218	220.9	458.48	**188**	**188.3**	**193.96**
800_800	2	**401**	505	506.6	397.95	**401**	**401**	**0.4**
800_1000	2	**384**	451	452.4	402.45	386	386.1	**40.66**
800_2000	4	<=480	531	533.2	366.94	**478**	**478.5**	**185.76**
800_5000	8	<=451	527	531.7	428.24	**450**	**452.5**	**117.54**
800_10000	11	<=363	405	409.3	399.02	**340**	**341**	**309.17**
1000_1000	2	**501**	631	635.3	441.81	**501**	**501**	**0.84**
1000_5000	6	<=528	610	612.8	492.49	**519**	**519.6**	**399.23**
1000_10000	9	<=450	520	521.6	291.92	**433**	**433.4**	**248.69**
1000_15000	13	<=463	506	507.9	363.71	**426**	**428.8**	**420.94**
1000_20000	16	<=440	469	473.3	276.97	**402**	**402.1**	**149.57**

sum of the costs of vertices (not in the D_k) whose Z_v are equal to k in the neighbor set of u and $cost(u)$.

V. VERTEX SELECTION STRATEGY

To improve the efficiency of the local search and forbid the cycling problem, we propose a vertex selection strategy by combining the scoring strategy, MKDSCC^2 strategy, random walk and tabu strategy. To implement tabu strategy, we employ a list named tabu_list to prevent removing the vertices which are just added in the last step. Specifically, the vertex selection strategy is based on the following four rules:

REMOVE-RULE1. For each vertex in D_k, select one vertex v with the greatest $score(v)$ value. If more than one exists, the vertex with the greatest value of $age[v]$ will be selected, and then update the MKDSCC^2 values of this vertex and its neighbors.

REMOVE-RULE2. For each vertex in D_k, select one vertex v which is not in tabu_list with the greatest $score(v)$ value. If more than one exists, the vertex with the greatest value of $age[v]$ will be selected, and then update the MKDSCC^2 values of this vertex and its neighbors.

ADD-RULE1. For each vertex not in D_k, select one vertex v randomly. And then update the MKDSCC^2 values of this vertex and its neighbors.

ADD-RULE2. For each vertex not in D_k with MKDSCC$^2[v] = 1$, select one vertex v with the greatest $score(v)$ value. If more than one exists, the vertex with the greatest value of $age[v]$ will be selected, and then update the MKDSCC^2 values of this vertex and its neighbors.

In detail, when VSCC^2 finds a solution, it removes a vertex from the solution and continues to search for a better k-dominating set. In this phase of removing vertices, we use REMOVE-RULE1 to remove a vertex from D_k. During the search for a solution, VSCC^2 swaps some vertices, i.e., removing one vertex from D_k according to REMOVE-RULE2 and then iteratively adding vertices into D_k. In the process of adding vertices, to increase the diversity of search, our algorithm selects one vertex according to ADD-RULE1 with probability p, or selects one vertex according to ADD-RULE2 with probability 1- p.

VI. VSCC^2 ALGORITHM

In this section, our algorithm framework VSCC^2 is proposed by integrating these strategies discussed above. The pseudo code of VSCC^2 is displayed as Algorithm 1.

At first, preprocessing is very necessary for the minimum k-dominating set. There are some vertices whose degrees are less than k in the graph, then these vertices must be added into in the candidate solution D_k. And in this case, we mark such vertices so that they will not be removed from the candidate solution D_k during the local search.

In the initialization process, VSCC^2 initializes *tabu_list*, *MKDSCC*^{2}, and the *cost* and *score* of vertices. Then the candidate solution D_k is constructed greedily by iteratively picking one vertex with the greatest *score*. The best solution D_k^* is initialized as current solution D_k.

After initialization, the main outer loop from lines 5 to 23 is executed until stop criterion is satisfied. When a better solution is obtained, the algorithm updates the D_k^*. After then, our algorithm chooses one vertex in D_k and removes it according to REMOVE-RULE1 until D_k is not a k-dominating set. At the same time, the MKDSCC^2 array should be updated according to MKDSCC^2-RULE2.

Then our algorithm selects one vertex and removes it from D_k according to REMOVE-RULE2. After removing a vertex, VSCC^2 updates MKDSCC^2 array according to MKDSCC^2-RULE2. The inner loop is from lines 14 to 22 until a k-dominating set is constructed. To increase the diversity of the search, our algorithm proposes a valid random walk strategy. In detail, VSCC^2 selects one vertex according to ADD-RULE1 with probability p, or selects one vertex according to ADD-RULE2 with probability 1- p. When the

TABLE 5. The comparative results of CPLEX, GRASP, and VSCC^2 with k_{max} on the general graphs benchmarks.

Instance	k_{max}	CPLEX	GRASP			VSCC^2		
			Min	Avg	Avgtime	Min	Avg	Avgtime
50_50	2	**26**	**26**	**26**	7.26	**26**	**26**	**0**
50_100	4	**37**	**37**	**37**	5.5	**37**	**37**	**0**
50_250	9	**36**	37	37	116.2	**36**	**36**	**0**
50_500	13	**29**	31	31.7	167.32	**29**	**29.3**	**193.13**
50_750	18	**27**	28	28.6	184.46	28	28	**0.01**
50_1000	22	**26**	**26**	26.9	3.34	**26**	**26**	**0.01**
100_100	2	**51**	52	52	57.05	**51**	**51**	**0.01**
100_250	6	**83**	**83**	**83**	0.28	**83**	**83**	**0**
100_500	9	**70**	73	73.9	109.8	**70**	**70**	**0.07**
100_750	13	**72**	74	74.5	291.4	**72**	**72**	**0.05**
100_1000	16	**68**	72	72.8	210.83	**68**	**68**	**0.9**
100_2000	25	<=**58**	63	63.8	178.73	**58**	**58.8**	**123.17**
150_150	2	**76**	87	87.7	476.99	**76**	**76**	**0.02**
150_250	4	**119**	**119**	**119**	0.71	**119**	**119**	**0**
150_500	6	**102**	104	104.5	286.97	**102**	**102**	**0.03**
150_750	9	**106**	111	111.4	263.79	**106**	**106**	**0.03**
150_1000	12	**108**	114	114.5	218.05	**108**	**108**	**24.99**
150_2000	21	<=**104**	114	115.5	308.79	**104**	**104**	**122.89**
150_3000	29	<=**100**	110	110.5	253.46	**100**	**100**	**97.76**
200_250	3	**156**	**156**	**156**	43.88	**156**	**156**	**0.01**
200_500	5	**141**	145	145.4	509.62	**141**	**141**	**0.03**
200_750	7	**142**	150	151.2	235.91	**142**	**142**	**0.13**
200_1000	11	**163**	166	166.6	365.07	**163**	**163**	**0.02**
200_2000	17	<=145	154	155.1	559.64	**143**	**143**	**19.63**
200_3000	23	<=137	154	154.1	399.96	**136**	**136.9**	**4.97**
250_250	2	**126**	151	151.2	389.31	**126**	**126**	**0.13**
250_500	4	**174**	180	180.7	359.73	**174**	**174**	**0.05**
250_750	6	**183**	189	190.5	298.81	**183**	**183**	**0.09**
250_1000	8	**186**	195	195.9	418.5	**186**	**186**	**1.03**
250_2000	14	<=184	195	196.6	413.7	**180**	**180**	**3.15**
250_3000	19	<=176	194	195.1	300.82	**173**	**173.1**	**202.74**
250_5000	30	<=174	191	191.9	366.04	**173**	**173.4**	**207.34**
300_300	2	**151**	182	183	281	**151**	**151**	**0.14**
300_500	4	**239**	242	242.4	434.5	**239**	**239**	**0.01**
300_750	5	**215**	224	224.5	167.84	**215**	**215**	**0.08**
300_1000	7	**229**	238	238.7	224.89	**229**	**229**	**0.05**
300_2000	14	**244**	252	252.7	303.94	**244**	**244**	**0.29**
300_3000	17	<=217	238	238.3	262.75	**214**	**214.8**	**1.78**
300_5000	25	<=207	230	231.4	583.34	**202**	**205.1**	**55.95**
500_500	2	**251**	311	312	397.42	**251**	**251**	**0.55**
500_1000	5	**412**	418	418.7	472.08	**412**	**412**	**0.15**

TABLE 5. ***(Continued.)*** **The comparative results of CPLEX, GRASP, and VSCC^2 with k_{max} on the general graphs benchmarks.**

500_2000	8	<=376	393	394.1	543.19	**374**	**374**	**1.73**
500_5000	17	<=365	396	397.6	525.1	**356**	**357.5**	**45.04**
500_10000	29	<=341	386	386.6	452.2	**337**	**338.7**	**177.63**
800_800	1	**267**	328	329.2	377.12	**267**	**267**	**8.07**
800_1000	3	**636**	645	646.5	297.06	**636**	**636**	**0.12**
800_2000	6	**646**	661	662.9	284.86	**646**	**646**	**0.15**
800_5000	13	<=640	672	673.5	351.32	**637**	**637**	**11.76**
800_10000	20	<=570	643	643.8	317.27	**556**	**558.7**	**0.98**
1000_1000	2	**501**	631	635.3	437.28	**501**	**501**	**2.82**
1000_5000	10	<=770	815	815.8	373.37	**764**	**764.8**	**189.46**
1000_10000	17	<=738	808	813.8	352.1	**715**	**719.3**	**148.01**
1000_15000	23	<=701	802	803.4	517.85	**685**	**690.3**	**0.86**
1000_20000	30	<=717	804	809.8	760.4	**689**	**694.6**	**0.76**

TABLE 6. **The comparative results of CPLEX, GRASP, and VSCC^2 with k_{min} on UDG benchmarks.**

			GRASP			VSCC^2		
Instance	k_{min}	CPLEX	Min	Avg	AvgTime	Min	Avg	AvgTime
50_150	2	**23.1**	23.5	23.52	49.94	**23.1**	**23.1**	**0**
50_200	2	**17.5**	17.7	17.7	19.04	**17.5**	**17.5**	**0**
100_150	2	**31.5**	32.6	32.86	114.01	**31.5**	**31.5**	**0.06**
100_200	2	**19.9**	20.4	20.85	102.64	**19.9**	**19.9**	**0.01**
250_150	2	**34.3**	38	38.75	259.6	**34.3**	**34.3**	**3.26**
250_200	2	**21.3**	23.5	23.99	226.12	**21.3**	**21.3**	**0.55**
500_150	2	**<=35.9**	42.6	43.23	314.08	**35.9**	**35.96**	**34.08**
500_200	2	**<=22**	26.2	26.8	198.86	**22**	**22**	**2.35**
800_150	2	**36.8**	45.9	46.43	341.92	36.9	36.95	**110.04**
800_200	2	**22.3**	27.6	28.16	243.17	**22.3**	**22.3**	**31.46**
1000_150	2	**37**	47	47.64	304.69	37.2	37.42	**114.07**
1000_200	2	**22.8**	28.2	28.65	283.77	**22.8**	**22.8**	**7.82**

selected vertex is added into the D_k, we need to update the MKDSCC^2 array according to MKDSCC^2-RULE3 and this vertex is added into *tabu_list*. After adding a vertex each time, the cost of each non-k-dominated vertex is increased by one. When reaching the stop criterion, the best solution of minimum k-dominating set problem will be returned.

VII. EXPERIMENTAL EVALUATION

In this section, we carry out extensive experiments to evaluate VSCC^2 on standard benchmarks. At present, there are few heuristic algorithms for MKDSP in the literatures as we know, thus the experimental results of VSCC^2 are compared with those of CPLEX, which is a high-performance solver for linear and mixed integer linear programs. To further test the effectiveness of VSCC^2, we also implement a classic GRASP algorithm, which is widely used in solving combinatorial optimization problems. And the experimental results of VSCC^2 are also compared with those of GRASP.

In our experiments, there are three classic benchmark instances, general graphs which are got from Type1 instance in [33], unit disk graphs (UDG) which are created by using the topology generator in [34] and DIMACS which is downloaded from http://iridia.ulb.ac.be/∼fmascia/ maximum _clique/. For general graphs and UDG, the number of vertices varies from 50 to 1000. In the case of general graphs, the number of edges varies from 50 to 20000. And in the case of UDG, there are two transmission ranges of 150 and 200 units. The size of DIMACS ranges from 150 vertices and

TABLE 7. The comparative results of CPLEX, GRASP, and VSCC^2 with k_{med} on UDG benchmarks.

			GRASP			VSCC^2		
Instance	k_{med}	CPLEX	Min	Avg	Avgtime	Min	Avg	Avgtime
50_150	2.9	**31.6**	32	32	0.12	**31.6**	**31.6**	**0**
50_200	3.5	**27.7**	28	28.16	89.44	**27.7**	**27.7**	**0**
100_150	4.2	**59.4**	61.2	61.67	190.87	**59.4**	**59.4**	**0.01**
100_200	5.8	**51.3**	54.5	55.02	233.19	**51.3**	**51.3**	**0.04**
250_150	7.8	<=123.5	136	136.9	269.93	**113.1**	**113.58**	**51.51**
250_200	11.6	<=115.1	128.1	129.34	361.61	**112.6**	**113.01**	**137.87**
500_150	13.6	<=235.4	260.8	261.99	349.3	**224.2**	**225**	**215.84**
500_200	21.1	<=223.2	254.9	257.47	442.75	**215.4**	**216.07**	**241.23**
800_150	20	<=353.2	403.9	406.45	451.82	**339.6**	**340.23**	**244.45**
800_200	32.4	<=347	411.5	413.86	437.71	**328**	**339**	**352.05**
1000_150	24	<=431.1	493.4	495.69	464.86	**411.7**	**412.89**	**275.06**
1000_200	38.7	<=421.1	500.5	504.5	446.56	**408.5**	**409.87**	**416.49**

TABLE 8. The comparative results of CPLEX, GRASP, and VSCC^2 with k_{max} on UDG benchmarks.

			GRASP			VSCC^2		
Instance	k_{max}	CPLEX	Min	Avg	Avgtime	Min	Avg	Avgtime
50_150	3.6	**36.5**	36.8	36.8	3.22	**36.5**	**36.6**	**0**
50_200	5.2	**37.4**	37.9	37.99	17.42	**37.4**	**37.4**	**0**
100_150	6.6	**79.8**	81	81.12	55.72	**79.8**	**79.8**	**0.01**
100_200	9.6	**74.7**	76.9	77.35	142.21	**74.7**	**74.7**	**0.01**
250_150	13.5	<=188.1	199.6	200.54	285.06	**187.1**	**187.16**	**62.77**
250_200	21.4	<=184.6	198.1	198.94	370.27	**182.1**	**182.34**	**27.56**
500_150	25.4	<=376.1	403.2	405	412.3	**368.4**	**368.53**	**215.79**
500_200	40.2	<=363.3	399.6	401.34	422.52	**357.1**	**358.14**	**187.29**
800_150	37.7	<=578.4	632.5	634.99	448.44	**568.4**	**568.95**	**274.74**
800_200	62.7	<=574.9	644.9	647.14	421.29	**570**	**571.99**	**189.18**
1000_150	46	<=714.8	785.9	788.13	497.36	**701.2**	**702.76**	**211.25**
1000_200	75.5	<=703.7	803.8	807.13	391.33	**699.4**	**703.01**	**123.65**

300 edges to over 4000 vertices and 7900000 edges. For the minimum k-dominating set problem, the different values of k for each instance are presented in Table 1.

Our algorithm VSCC^2 and GRASP are programmed in C++ and compiled by g++ with the -O2 option on the Linux Ubuntu with 2.3GHZ and 8 GB. For each instance, VSCC^2 and GRASP perform ten independent runs with different random seeds, which one run is stopped until arriving at a time limit. In this paper, the time limit is set to 1000s for general graphs and UDG, otherwise the time limit is set to 1800s. And the termination condition of CPLEX is 3600s. The parameter p value is determined by performing a preliminary experiment, the different p values for each instance are presented in Table 2.

In the results of our experiment, we denote the best solution values (Min), average solution values (Avg), and the average run time to reach the best solution (AvgTime, in seconds). It is worthy to note that the bold value presents the best solution value or the shortest run time among the different algorithms compared. For some instances, CPLEX is not capable to find a k-dominating set, then it is marked as "n/a" for these cases. And "<=" denotes that CPLEX finds the upper bound of instances.

A. RESULTS ON GENERAL GRAPH BENCHMARKS

The performance results of algorithms with k_{min}, k_{med}, k_{max} on the general graphs are displayed in Tables 3-5. From three tables, we observe that VSCC^2 is faster than GRASP

TABLE 9. **The comparative results of CPLEX, GRASP, and VSCC^2 with k_{min} on DIMACS benchmarks.**

			GRASP			VSCC^2		
Instance	k_{min}	CPLEX	Min	Avg	Avgtime	Min	Avg	Avgtime
brock200_2	2	**6**	**6**	**6**	7.659	**6**	**6**	**0.02**
brock200_4	2	**9**	**9**	**9**	12.215	**9**	**9**	**0.02**
brock400_2	2	<=14	14	14	4.924	**13**	**13**	**297.37**
brock400_4	2	<=14	14	14	15.104	**13**	**13**	**88.65**
brock800_2	2	<=12	12	12	4.376	**11**	**11**	**446.46**
brock800_4	2	<=12	12	12	2.492	**11**	**11**	**176.53**
C125.9	2	**22**	23	23	6.132	**22**	**22**	**0.04**
C250.9	2	<=26	26	26	226.12	**25**	**25**	**0.21**
C500.9	2	<=31	31	31.7	336.65	**30**	**30**	**3.66**
C1000.9	2	<=37	36	36.9	122.31	**35**	**35**	**216.11**
C2000.5	2	<=10	**9**	9.9	97.07	**9**	**9**	**443.42**
C2000.9	2	<=43	43	43	297.03	**41**	**41.9**	**104.23**
C4000.5	2	n/a	11	11	18.04	**10**	**10.8**	**212.14**
DSJC500.5	2	<=8	**7**	7.9	83.88	**7**	**7**	**5.7**
DSJ1000.5	2	<=9	**8**	**8**	640.99	**8**	**8**	**29.72**
gen200_p0.9_44	2	**<=25**	26	26	44.94	**25**	**25**	**0.05**
gen200_p0.9_55	2	**<=25**	**25**	**25**	214.63	**25**	**25**	**0.02**
gen400_p0.9_55	2	<=29	30	30	107.78	**28**	**28**	**1.34**
gen400_p0.9_65	2	<=30	29	29.5	384.97	**28**	**28**	**40.17**
gen400_p0.9_75	2	**<=29**	30	30	513.7	**29**	**29**	**0.23**
hamming8-4	2	**8**	**8**	**8**	0.02	**8**	**8**	**0.01**
hamming10-4	2	**<=18**	20	20.7	288.99	**18**	**18**	**27.34**
keller4	2	**7**	8	8	4.13	**7**	**7**	**0.01**
keller5	2	<=14	14	14.9	29.49	**13**	**13**	**34.98**
keller6	2	n/a	25	25	35.19	**22**	**22.6**	**680.47**
MANN_a27	2	**142**	143	143.6	231.98	**142**	**142**	**63.71**
MANN_a45	2	**<=373**	374	374	435.62	374	374	**6.27**
MANN_a81	2	**<=1159**	1161	1161	34.85	1161	1161	**0.08**
p_hat300-1	2	**4**	**4**	**4**	4.59	**4**	**4**	**0.05**
p_hat300-2	2	**6**	**6**	**6**	142.76	**6**	**6**	**0.01**
p_hat300-3	2	**11**	13	13	37.51	**11**	**11**	**0.1**
p_hat700-1	2	**4**	5	5	7.59	**4**	**4**	**0.24**
p_hat700-2	2	<=7	8	8	151.31	**6**	**6**	**0.4**
p_hat700-3	2	<=14	17	17	423.12	**13**	**13**	**0.38**
p_hat1500-1	2	**<=5**	**5**	5.7	292.45	**5**	**5**	**0.48**
p_hat1500-2	2	<=9	9	9.6	602.84	**7**	**7**	**1.09**
p_hat1500-3	2	<=16	20	20.6	676.9	**15**	**15**	**2.3**

on all instances. In Table 3, we can observe that our algorithm VSCC^2 can obtain better solution values than GRASP for 45 instances, and the same solution values for the rest 9 instances. Furthermore, we find that the results of our algorithm VSCC^2 are equal to the optimal solutions of CPLEX for 31 out of 54 instances, and our algorithm VSCC^2 can

TABLE 10. The comparative results of CPLEX, GRASP, and VSCC^2 with k_{med} on DIMACS benchmarks.

			GRASP			VSCC^2		
Instance	k_{med}	CPLEX	Min	Avg	Avgtime	Min	Avg	Avgtime
brock200_2	31	<=65	65	65	374.01	**62**	**62**	**8.91**
brock200_4	23	<=70	74	74.2	620.63	**67**	**67.9**	**143.64**
brock400_2	32	<=139	138	138.7	485	**129**	**129**	**429.32**
brock400_4	32	<=137	139	139.5	421.47	**129**	**129.7**	**356.4**
brock800_2	83	<=261	258	258.3	512.74	**244**	**244.6**	**310.27**
brock800_4	81	<=253	251	251.9	402.42	**238**	**238.5**	**208.43**
C125.9	7	<=**63**	69	69.7	280.88	64	64	**0.12**
C250.9	13	<=125	139	140	731.26	**123**	**123.5**	**388.67**
C500.9	18	<=193	198	199	584.04	**182**	**182.7**	**465.32**
C1000.9	34	<=386	381	382.4	289.28	**351**	**351.8**	**846.63**
C2000.5	271	<=581	577	577.2	588.83	**557**	**558**	**1047.59**
C2000.9	63	<=744	697	697.4	433.9	**654**	**654.2**	**1004.82**
C4000.5	527	n/a	1107	1108.4	1081.71	**1082**	**1083.7**	**28.95**
DSJC500.5	71	<=153	152	153	393.36	**145**	**145.8**	**160.53**
DSJ1000.5	139	<=301	298	298	755.95	**286**	**286**	**532.82**
gen200_p0.9_44	10	<=98	111	112.6	568.43	**95**	**95.8**	**132.5**
gen200_p0.9_55	10	<=99	109	109.9	914.56	**96**	**96**	**47.49**
gen400_p0.9_55	17	<=**176**	204	204.2	438.92	179	180.1	**248**
gen400_p0.9_65	18	<=186	208	211.3	546.71	**185**	**185.9**	**243.24**
gen400_p0.9_75	17	<=**175**	204	205.2	857.49	178	178	**153.63**
hamming8-4	24	<=68	69	69	386.57	**67**	**67.3**	**496.21**
hamming10-4	45	<=304	285	286.1	918.95	**271**	**271**	**161.29**
keller4	18	<=53	56	56	689.79	**51**	**51**	**141.69**
keller5	55	<=245	248	248.5	379.29	**229**	**229.7**	**597.23**
keller6	169	n/a	1053	1056.4	1053.36	**961**	**961.6**	**765.81**
MANN_a27	4	**351**	**351**	**351**	645.87	**351**	**351**	**20.18**
MANN_a45	7	**990**	**990**	**990**	403.91	**990**	**990**	**8.23**
MANN_a81	11	**3240**	**3240**	**3240**	282.01	**3240**	**3240**	**0.02**
p_hat300-1	70	<=**96**	103	103.8	344.09	**96**	**96**	**438.42**
p_hat300-2	61	<=**148**	159	159.3	340.81	149	150.7	**332.67**
p_hat300-3	34	<=**157**	179	179.9	490.24	159	160.2	**137.58**
p_hat700-1	157	<=222	239	239.8	746.3	**219**	**219**	**3.98**
p_hat700-2	137	<=**350**	386	387.2	871.23	354	362.4	**1.39**
p_hat700-3	74	<=**365**	431	433.3	537.33	372	375.6	**0.63**
p_hat1500-1	337	n/a	506	511.2	934.13	**472**	**472**	**236.13**
p_hat1500-2	292	<=802	818	820.4	918.17	**765**	**776.6**	**68.56**
p_hat1500-3	148	<=797	854	856.6	726.98	**780**	**783.6**	**3.48**

reach or improve the upper bound of CPLEX for 20 out of 54 instances. In Table 4, we can observe that VSCC^2 also performs better than GRASP. It is encouraging to see that VSCC^2 outperforms CPLEX on all the 54 instances except for 800_1000. In Table 5, we can observe that VSCC^2 performs better than GRASP in 48 out of 54 instances. Compared to CPLEX, VSCC^2 obtains better results for all the instances with one exception, i.e. 50_750.

TABLE 11. The comparative results of CPLEX, GRASP, and VSCC^2 with k_{max} on DIMACS benchmarks.

			GRASP			VSCC^2		
Instance	k_{max}	CPLEX	Min	Avg	Avgtime	Min	Avg	Avgtime
brock200_2	61	<=**119**	123	123.7	368.61	**119**	**119**	**11.32**
brock200_4	44	<=**124**	133	133.9	373.31	**124**	**124**	**217.64**
brock400_2	63	<=250	262	264	447.52	**246**	**246**	**3.48**
brock400_4	62	<=244	261	262.1	637.25	**242**	**242.9**	**3.98**
brock800_2	164	<=487	492	493.9	937.48	**472**	**472.5**	**191.99**
brock800_4	159	<=472	477	478.9	364.06	**456**	**456.9**	**53.47**
C125.9	11	**88**	93	93.9	466.78	**88**	**88**	**0.07**
C250.9	23	<=194	201	202.5	893.62	**192**	**192**	**1.09**
C500.9	34	<=328	362	363.3	630.35	**321**	**322.3**	**0.66**
C1000.9	66	<=665	720	720.9	940.59	**645**	**650.6**	**1.46**
C2000.5	540	<=1116	1124	1126.4	994.15	**1090**	**1091.2**	**608.45**
C2000.9	124	<=1291	1339	1339.8	1163.91	**1234**	**1242.1**	**64.43**
C4000.5	1052	n/a	2177	2178.3	1014.08	**2125**	**2126.4**	**158.73**
DSJC500.5	140	<=289	296	296.9	442.12	**282**	**282**	**359.96**
DSJ1000.5	276	<=571	579	579.2	548.49	**557**	**557**	**315.13**
gen200_p0.9_44	17	<=**144**	156	156.5	349.87	**144**	**144**	**5.24**
gen200_p0.9_55	18	<=149	158	158.9	565.82	**148**	**148**	**53.84**
gen400_p0.9_55	33	<=298	328	328.8	862.88	**292**	**293.6**	**22.09**
gen400_p0.9_65	33	<=294	322	323.3	531.88	**287**	**287**	**46.79**
gen400_p0.9_75	32	<=283	322	323.1	820.89	**279**	**280.8**	**0.43**
hamming8-4	46	<=128	128	134.1	825.98	**125**	**125.2**	**656.58**
hamming10-4	88	<=512	542	546.8	785.54	**510**	**510.9**	**740.17**
keller4	34	<=94	102	102.7	354.45	**93**	**93.3**	**17.86**
keller5	108	<=433	447	449	989.28	**424**	**425.2**	**268.26**
keller6	335	n/a	1918	1920.7	713.31	**1810**	**1818.3**	**6.26**
MANN_a27	7	**351**	**351**	**351**	671.29	**351**	**351**	**17.83**
MANN_a45	11	**990**	**990**	**990**	365.83	**990**	**990**	**3.73**
MANN_a81	20	**3240**	**3240**	**3240**	291.08	**3240**	**3240**	**0.03**
p_hat300-1	138	<=**186**	208	208.1	682.12	193	194.2	**52.14**
p_hat300-2	120	**214**	268	269.7	519.19	216	217.7	**0.34**
p_hat300-3	66	<=**225**	264	264.3	584.84	226	226.4	**0.21**
p_hat700-1	312	<=**432**	483	483.6	598.38	445	448.5	**0.76**
p_hat700-2	271	<=**500**	656	658.9	805.89	501	510.3	**2.46**
p_hat700-3	146	<=**517**	642	643.5	790.3	518	521	**0.67**
p_hat1500-1	671	n/a	1042	1043.7	807.83	**970**	**987**	**12.88**
p_hat1500-2	582	<=1500	1444	1444.9	999.58	**1100**	**1119.7**	**13.29**
p_hat1500-3	294	<=1092	1424	1425.9	1069.59	**1086**	**1093.8**	**7.58**

B. RESULTS ON UDG BENCHMARKS

The experimental results of algorithms with k_{min}, k_{med}, k_{max} on the UDG graphs are presented in Tables 6-8. There are 12 groups of instances for the UDG graphs, each of which contains 10 instances. From Table 6, we can observe that the quality of solutions found by VSCC^2 is much better than those found by GRASP. For two instances, i.e. 800_150 and 1000_150, CPLEX is much better than VSCC^2 in terms of

Algorithm 1 VSCC^2 ()

preprocess the instance;
initialize *tabu_list*, *MKDSCC*^{2}, and the *cost* and *score* of vertices;
initialize the candidate solution D_k greedily;
$D_k^* := D_k$;
while stop criterion is not satisfied **do**
  while D_k is a k-dominating set **then**
    if $m(D_k) < m(D_k^*)$ **then** $D_k^* := D_k$;
    $v :=$ pick x according to **REMOVE-RULE1**;
    $D_k := D_k \backslash \{v\}$ and update *MKDSCC*^{2} according to **MKDSCC^2-RULE2**;
  end while
  $v :=$ pick x according **to REMOVE-RULE2**;
  $D_k := D_k \backslash \{v\}$ and update *MKDSCC*^{2} according to **MKDSCC^2-RULE2**;
  tabu_list $:= \emptyset$;
  while there are non-k-dominated vertices **do**
    if rand(0,1) $< p$
      $v :=$ pick x according to **ADD-RULE1**;
    else
      $v :=$ pick x according to **ADD-RULE2**;
    $D_k := D_k \cup \{v\}$ and update *MKDSCC*^{2} according to **MKDSCC^2-RULE3**;
    tabu_list $:=$ *tabu_list* $\cup \{v\}$;
    $cost(u) := cost(u) + 1$, for each non-k- dominated vertex;
  end while
end while
return D_k^*;

the quality of solutions. As is clear from Table 7, VSCC^2 with k_{med} shows significant superiority on all instances. In Table 8, we can observe that VSCC^2 with k_{max} outperforms GRASP and CPLEX in all instances.

C. RESULTS ON DIMACS BENCHMARKS

Tables 9-11 summarize the computational results of the algorithms with k_{min}, k_{med}, k_{max} on DIMACS benchmarks. For C4000.5 and Keller6, CPLEX is not capable to find a k-dominating set with k_{min}, k_{med}, k_{max}. For p_hat1500-1, CPLEX is not capable to find a k-dominating set with k_{med}, k_{max}. For these three instances, GRASP and VSCC^2 can find feasible solutions and VSCC^2 performs much better than GRASP. And VSCC^2 outperforms GRASP algorithm on almost all instances. In Table 9, both CPLEX and VSCC^2 can find the 10 optimal solutions for all 37 instances. For 25 instances, the results of VSCC^2 can reach or improve the upper bound of CPLEX. From Tables 10, 11, we find that the results of our algorithm VSCC^2 are equal to the optimal solutions of CPLEX for 3,4 out of 37 instances respectively. And VSCC^2 can reach or improve the upper bound of CPLEX for 27 out of 37 instances in both Table 10 and Table 11. As can be seen in three tables, VSCC^2 is faster than GRASP on the all instances of DIMACS.

VIII. CONCLUSION

In this paper, we develop a new local search algorithm VSCC^2 for the MKDS problem. The minimum k-dominating set two-level configuration checking (MKDSCC^2) strategy is used to alleviate the cycling problem in the local search. Combing MKDSCC^2, random walk, with the scoring strategy, a vertex selection strategy is proposed to decide which vertex should be added into or removed from the candidate solution. We assess the performance of the VSCC^2 algorithm on the 211 classical instances with different values of k. The results show that the VSCC^2 algorithm outperforms CPLEX and GRASP on almost all instances. Finally, these ideas can be beneficially applied to other combinatorial problems because those are mentioned in the introduction of these work [35]–[38].

ACKNOWLEDGMENTS

The authors are grateful to all anonymous reviewers whose insightful comments have helped us to improve the work.

REFERENCES

[1] Y. Alavi, G. Chartrand, L. Lesniak-Foster, D. R. Lick, and C. E. Wall, *Graph Theory With Applications to Algorithms and Computer Science*. New York, NY, USA: Wiley, 1985.
[2] H. Samuel, W. Zhuang, and B. Preiss, "DTN based dominating set routing for MANET in heterogeneous wireless networking," *Mobile Netw. Appl.*, vol. 14, no. 2, pp. 154–164, 2009.
[3] J. M. Gorce *et al.*, "Optimization in wireless networks," *Cancer Res.*, vol. 24, no. 2, p. 1473, 1964.
[4] B. Pazand and A. Datta, "Minimum dominating sets for solving the coverage problem in wireless sensor networks," in *Proc. Int. Conf. Ubiquitous Comput. Syst.*, vol. 4239. New York, NY, USA: Springer-Verlag, 2006, pp. 454–466.
[5] C. Shen and T. Li, "Multi-document summarization via the minimum dominating set," in *Proc. 23rd Int. Conf. Comput. Linguistics (COLING)*, 2010, pp. 984–992.
[6] Y.-Z. Xu and H.-J. Zhou, "Generalized minimum dominating set and application in automatic text summarization," in *Proc. Int. Meeting High-Dimensional Data-Driven Sci.*, vol. 699, 2016, p. 012014.
[7] T. N. Dinh, Y. Shen, D. T. Nguyen, and M. T. Thai, "On the approximability of positive influence dominating set in social networks," *J. Combinat. Optim.*, vol. 27, no. 3, pp. 487–503, 2014.
[8] L. H. Yen and Z. L. Chen, "Self-stabilizing distributed formation of minimal k-dominating sets in mobile ad hoc networks," in *Proc. 10th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, vol. 28, no. 1, Aug. 2014, pp. 723–728.
[9] C. Mathieu, M. Barbeau, P. Bose, and E. Kranakis, "Incremental construction of k-dominating sets in wireless sensor networks," in *Principles of Distributed Systems*. Berlin, Germany: Springer, 2006, pp. 202–214.
[10] G. R. Argiroffo, M. E. Ugarte, and M. S. Escalante, "On the k-dominating set polytope of Web graphs," *Electron. Notes Discrete Math.*, vol. 36, no. 36, pp. 1161–1168, 2010.
[11] A. Hansberg and R. Pepper, "On k-domination and j-independence in graphs," *Graphs Combinat.*, vol. 28, pp. 1–55, 2013.
[12] N. Meddah and M. Blidia, "Vertices contained in all or in no minimum k-dominating sets of a tree," *AKCE Int. J. Graphs Combinat.*, vol. 11, no. 1, pp. 105–113, 2014.
[13] H. A. Harutyunyan and A. L. Liestman, "Upper bounds on the broadcast function using minimum dominating sets," *Discrete Math.*, vol. 312, no. 20, pp. 2992–2996, 2012.
[14] J. M. M. vam Rooij and H. L. Bodlaender, "Exact algorithms for dominating set," *Discrete Appl. Math.*, vol. 159, pp. 2147–2164, Oct. 2011.

[15] F. V. Fomin, D. Kratsch, and G. J. Woeginger, "Exact (exponential) algorithms for the dominating set problem," in *Proc. Int. Workshop Graph-Theoretic Concepts Comput. Sci.* New York, NY, USA: Springer-Verlag, 2004, pp. 245–256.
[16] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov, "Bounding the number of minimal dominating sets: A measure and conquer approach," in *Proc. Int. Conf. Algorithms Comput.* New York, NY, USA: Springer-Verlag, 2005, pp. 573–582.
[17] A.-R. Hedar and R. Ismail, "Hybrid genetic algorithm for minimum dominating set problem," in *Proc. Int. Conf. Comput. Sci. Appl.*, Fukuoka, Japan, 2010, pp. 457–467.
[18] C. N. Giap and D. T. Ha, "Parallel genetic algorithm for minimum dominating set problem," in *Proc. Int. Conf. Comput., Manage. Telecommun.*, Apr. 2014, pp. 165–169.
[19] D. Chalupa, "An order-based algorithm for minimum dominating set with application in graph mining," *Inf. Sci.*, vol. 426, pp. 101–116, Feb. 2018.
[20] A.-R. Hedar and R. Ismail, "Simulated annealing with stochastic local search for minimum dominating set problem," *Int. J. Mach. Learn. Cybern.*, vol. 3, no. 2, pp. 97–109, 2012.
[21] Y. Wang, S. Cai, and M. Yin, "Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function," *J. Artif. Intell. Res.*, vol. 58, pp. 267–295, Feb. 2017.
[22] R. Li, S. Hu, P. Zhao, Y. Zhou, and M. Yin, "A novel local search algorithm for the minimum capacitated dominating set," *J. Oper. Res. Soc.*, vol. 3, pp. 1–16, Jul. 2017.
[23] S. Cai and K. Su, "Local search with configuration checking for SAT," in *Proc. IEEE Int. Conf. Tools Artif. Intell.*, Nov. 2011, pp. 59–66.
[24] S. Cai and K. Su, "Configuration checking with aspiration in local search for SAT," in *Proc. AAAI*, 2012, pp. 434–440.
[25] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su, "CCLS: An efficient local search algorithm for weighted maximum satisfiability," *IEEE Trans. Comput.*, vol. 64, no. 7, pp. 1830–1843, Jul. 2015.
[26] C. Luo, S. Cai, K. Su, and W. Wu, "Clause states based configuration checking in local search for satisfiability," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 1028–1041, May 2015.
[27] Y. Wang, D. Ouyang, L. Zhang, and M. Yin, "A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity," *Sci. China Inf. Sci.*, vol. 60, no. 6, p. 062103, 2017.
[28] R. Li, X. Wu, H. Liu, J. Wu, and M. Yin, "An efficient local search for the maximum edge weighted clique problem," *IEEE Access*, vol. 6, pp. 10743–10753, 2018.
[29] J. Ren, G. Zhang, and D. Li, "Multicast capacity for VANETs with directional antenna and delay constraint under random walk mobility model," *IEEE Access*, vol. 5, pp. 3958–3970, 2017.
[30] M. Xu, X. Zhou, Q. Huo, and H. Liu, "Efficient stochastic approximation Monte Carlo sampling for heterogeneous redundancy allocation problem," *IEEE Access*, vol. 4, pp. 7383–7390, 2016.
[31] S.-Y. Kuo and Y.-H. Chou, "Entanglement-enhanced quantum-inspired tabu search algorithm for function optimization," *IEEE Access*, vol. 5, pp. 13236–13252, 2017.
[32] W. P. Mathias-Neto and J. R. S. Mantovani, "A node-depth encoding-based tabu search algorithm for power distribution system restoration," *J. Control Automat. Elect. Syst.*, vol. 27, no. 3, pp. 317–327, 2016.
[33] R. Jovanovic, M. Tuba, and D. Simian, "Ant colony optimization applied to minimum weight dominating set problem," *Plant Physiol.*, vol. 146, no. 3, pp. 173–176, 2010.
[34] M. Mastrogiovanni. (2007). *The Clustering Simulation Framework: A Simple Manual*. [Online]. Available: http://www.michele-mastrogiovanni.Net/software/ download/README.pdf
[35] J. Jiang, D. Hao, Y. Chen, M. Parmar, and K. Li, "GDPC: Gravitation-based density peaks clustering algorithm," *Phys. A, Stat. Mech. Appl.*, vol. 502, pp. 345–355, Jul. 2018.
[36] R. Li, S. Cai, S. Hu, M. Yin, and J. Gao, "NuMWVC: A novel local search for minimum weighted vertex cover problem," in *Proc. AAAI*, 2018, pp. 8107–8108.
[37] R. Li, S. Hu, Y. Wang, and M. Yin, "A local search algorithm with tabu strategy and perturbation mechanism for generalized vertex cover problem," *Neural Comput. Appl.*, vol. 28, no. 7, pp. 1775–1785, 2017.
[38] Y. Wang and X. Li, "A hybrid chaotic biogeography based optimization for the sequence dependent setup times flowshop scheduling problem with weighted tardiness objective," *IEEE Access*, vol. 5, pp. 26046–26062, 2017.

RUIZHI LI received the B.S. and Ph.D. degrees in computer science from Northeast Normal University, China, in 2008 and 2017, respectively. She has been with the Jilin University of Finance and Economics. Her research interests include combinatorial optimization problem, heuristic search, and local search.

HUAN LIU received the B.S. degree in computer science from Northeast Normal University, China, in 2017, where she is currently pursuing the M.S. degree in computer science. Her research interests include local search and heuristic search.

XIAOLI WU received the B.S. degree in computer science from Northeast Normal University, China, in 2017, where she is currently pursuing the M.S. degree in computer science. Her research interests include exact algorithm and combinatorial optimization problem.

JUN WU was born in Wuhu, Anhui, in 1991. He received the B.S. degree from the College of Computer Sciences and Technology, Jilin University, in 2014. He is currently pursuing the master's degree with the College of Computer Sciences and Information Technology, Northeast Normal University. His research interests include automated reasoning, multi-objective optimization, intelligent algorithm, combinatorial optimization problem, local search, and heuristic search.

MINGHAO YIN received the B.S. and M.S. degrees from Northeast Normal University, China, in 2001 and 2004, respectively, and the Ph.D. degree from Jilin University, China, in 2008, all in computer science. He has been the Dean of the Department since 2010. He has authored two books and over 100 articles. His research interests include swarm intelligence, automated reasoning, automated planning, and algorithms.