Neural Comput & Applic (2017) 28 (Suppl 1):S143–S151
DOI 10.1007/s00521-016-2324-6

ORIGINAL ARTICLE

A path cost-based GRASP for minimum independent dominating set problem

Yiyuan Wang[1,2] · Ruizhi Li[2] · Yupeng Zhou[2] · Minghao Yin[2,3]

Received: 21 December 2015 / Accepted: 6 April 2016 / Published online: 26 April 2016

Abstract The minimum independent dominating set problem (MIDS) is an extension of the classical dominating set problem with wide applications. In this paper, we describe a greedy randomized adaptive search procedure (GRASP) with path cost heuristic for MIDS, as well as the classical tabu mechanism. Our novel GRASP algorithm makes better use of the vertex neighborhood information provided by path cost and thus is able to discover better and more solutions and to escape from local optimal solutions when the original GRASP fails to find new improved solutions. Moreover, to further overcome the serious cycling problem, the tabu mechanism is employed to forbid some just-removed vertices back to the candidate solution. Computational experiments carried out on standard benchmarks, namely DIMACS instances, show that our algorithm consistently outperforms two MIDS solvers as well as the original GRASP.

Keywords Minimum independent dominating set problem · GRASP · Path cost · Local search

✉ Minghao Yin
ymh@nenu.edu.cn

[1] College of Computer Science and Technology, Jilin University, Changchun, China

[2] School of Computer Science and Information Technology, Northeast Normal University, Changchun, China

[3] Key Laboratory of Symbol Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun, China

1 Introduction

Given an undirected graph $G = (V,E)$, a dominating set (DS) D is a subset of vertices such that each vertex not in D is adjacent to at least one member of D. An independent set (IS) I of G is a subset of vertices such that, for any $(v_i,v_j) \in I^*I$, $(v_i, v_j) \notin E$. An independent dominating set (IDS) in a given graph is a subset of vertices that is both dominating and independent. Equivalently, an independent dominating set is a maximum independent set. The minimum independent dominating set problem (MIDS) consists in identifying the smallest independent dominating set in this given graph.

MIDS has been widely used in various real-world domains. For example, a study of the survey of clustering algorithms for wireless networks [1, 2] shows that the initial clustering schemes for wireless networks are primarily IDS schemes such as in [3–5]. Renewed interest in IDS-based schemes is rising recently, in the context of wireless sensor networks (WSNs) [6, 7] and wireless sensor and actor networks (WSANs) [8, 9], which are useful in distributing actors and designing topologies that help to improve energy efficiency.

MIDS is known to be NP hard [10]. It is also very hard from an approximation view since there is no polynomial-time algorithm for these problems within ratio $|V|^{1-\varepsilon}$, for any $\varepsilon > 0$, unless P = NP [11]. Several approximation algorithms have been introduced to solve MIDS [12]. For MIDS, the trivial $O_*(2^{|V|})$ bound is initially broken by [13] down to $O_*(3^{|V|/3}) = O_*(2^{0.529|V|})$ using the result by [14], which means that the number of maximal (for inclusion) independent sets in a graph is at most $3^{|V|/3}$. This result has been dominated by [15] where an algorithm solving MIDS optimally through branch and reduce technique with running time $O_*(2^{0.441|V|})$ is proposed. For sparse graphs, e.g.,

graphs with degree bounded by 3 and 4, a new branching techniques can be applied to these graphs and the resulting algorithms [16] are with time complexities $O_*(2^{0.465|V|})$ and $O_*(2^{0.620|V|})$, respectively. In [17], a branching algorithm computing a MIDS in general graphs with running time $O_*(2^{0.424|V|})$ and polynomial space is designed.

Nevertheless, most problems [18–21] found in industry are either computationally intractable by their nature, or sufficiently large so as to preclude approximation algorithms. In such cases, heuristic methods [22–27] are usually employed to find good, but not necessarily guaranteed optimal solutions [28]. The effectiveness of these methods depends upon their ability to adapt to particular realization, avoid entrapment at local optima, and exploit the basic structure of the problem. However, there are relatively fewer heuristics for solving MIDS. In this paper, we develop a greedy randomized adaptive search procedure (GRASP) based on a new heuristic named path cost and a tabu mechanism for solving MIDS.

GRASP [29, 30] is a multi-start heuristic which consists of applying local search to feasible starting solutions generated with a greedy randomized construction heuristic. There are two phases including a construction phase and a local search phase, in each GRASP iteration. In the construction phase, a candidate solution is iteratively constructed by adding one element each time. The score of each element is adaptive, for it is associated with every element updated at each iteration of the construction phase to reflect the changes coming with the selection of the previous element. At each construction iteration, the next element to be added is randomly selected from the list of the best candidates, which is the highest score value. This list is also called the restricted candidate list (RCL), which means the selected element is not necessarily the top candidate. By applying RCL, different solutions are allowed to be obtained in the construction phase. However, the construction phase of GRASP is not guaranteed to obtain a local optimal with respect to simple adding operations. Hence, it should be beneficial to adapt a local search phase to search the neighborhood of the construction solution to obtain the improved solution. In the local search phase, a local optimum in the neighborhood of the constructed solution would be found. The local search phase works iteratively by swapping one element in the candidate solution with another one or by removing the redundant element from the candidate solution which is still the solution after being removed.

However, the effectiveness of a simple GRASP is not satisfying. The classical method to improve the GRASP process is called path relinking [31]. It is an intensification method that explores paths in the solution space connecting good-quality solutions. However, path relinking is not suitable for MIDS since an independent solution constructed by another solution is not very well. Therefore, we firstly propose a novel heuristic named path cost to improve the GRASP for MIDS. Each vertex will be associated with the path cost. In the construction phase, RCL is built by some vertices with best scores. The best score of each vertex is dynamically computed according to the path cost of vertices, while the previous score is calculated by the static method. In the local search phase, the selection of a swapping pair of vertices and a removed vertex are also depended on the path cost of each vertex. At each local search iteration, the value of path cost of uncovered vertex need to be increased. It requires the candidate solution to cover these vertices as soon as possible. On the other hand, at the end of local search phase, the path cost value of all vertices should be cleared to obtain the next construction solution.

The second heuristic is the tabu mechanism [32, 33] to overcome the cycling problem in the local search phase. Our algorithm uses the tabu mechanism to forbid reversing the recent changes, i.e., a swapping pair of vertices or a just-removed vertex, where the strength of forbidding is controlled by a parameter called tabu tenure. Combining with the path cost and the tabu mechanism, we develop a novel GRASP algorithm for MIDS, which is called GRASP + PC. We carry out extensive experiments to compare GRASP + PC with two MIDS solvers, namely CPLEX12.6 and LocalSolver5.5, on the classical DIMACS benchmark [34] from the Second DIMACS Implementation Challenge which introduces in real applications and randomly generated graphs. Experimental results show that GRASP + PC significantly outperforms CPLEX12.6 and LocalSolver5.5 on most instances.

The remainder of this paper is organized as follows: we give some necessary background knowledge in Sect. 2. Then, in Sect. 3 we propose a new path cost strategy followed by a novel GRASP + PC being proposed in Sect. 4. Section 5 gives the experimental evaluations and the analyses of the experimental result. Conclusions and future directions are introduced in the last section.

2 Background

An undirected graph $G = (V,E)$ comprises a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ of n vertices together with a set $E = \{e_1, e_2, \ldots, e_m\}$ of m edges, where each edge $e = (v,u)$ connects two vertices u and v where u is adjacent to v. The neighborhood of a vertex v is $N(v) = \{u \in V | (v,u) \in E\}$, and the close neighborhood of a vertex v is $N[v] = N(v) \cup \{v\}$.

The concepts of maximum independent set, minimum dominating set, minimum independent dominating set, and partial independent dominating set are defined as below.

Definition 1 (*Maximum independent set problem*) Given an undirected graph $G = (V,E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. The aim of *maximum independent set* problem is to find a maximum size subset *IS* of *V* such that no two vertices in *IS* are adjacent, i.e., $\forall v_i, v_j \in IS, (v_i, v_j) \notin E$.

Definition 2 (*Minimum dominating set problem*) Given an undirected graph $G = (V,E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. The task of *minimum dominating set* problem is to find a minimum size subset *DS* of *V* that has a property that every vertex in *V* either belongs to *DS* or is adjacent to a vertex in *DS*.

Definition 3 (*Minimum independent dominating set*) Given an undirected graph $G = (V,E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. The task of *minimum independent dominating set* problem is to find a minimum subset *IDS* of *V* such that each vertex of *V* not in *IDS* is adjacent to at least one member of *IDS* and no two vertices in *IDS* are adjacent.

Definition 4 (*Partial independent dominating set*) Given an undirected graph $G = (V,E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. V' is a subset of a set *V*. The task of *partial independent dominating set* problem is to find a subset *PIDS* of V' such that each vertex of V' not in *PIDS* is adjacent to at least one member of *PIDS and* no two vertices in *PIDS* are adjacent.

During the construction and local search phases of our algorithm, we need to maintain a partial independent dominating set as the candidate solution *CS*.

3 A novel path cost heuristic

During the construction and local search phases of our algorithm for MIDS, deciding which subset should be selected to be added into or removed from the candidate solution plays an important role in the efficiency of the search. In this section, we introduce a novel heuristic called path cost for each vertex during the vertices selection process. This heuristic can be used to decide which vertex could be picked as a candidate solution component.

Definition 5 (*Path cost*) Given an undirected graph $G = (V,E)$, for each vertex $v \in V$, we use a property path cost function, denoted by $pc(v)$, associated with v, which is maintained during the local search process.

In the initialization period of our algorithm, the $pc(v)$ is initialized as 1, for each vertex $v \in V$. To attain more solutions, we propose a forgetting cost strategy to improve the quality of solutions obtained by our algorithm. Firstly, we give two forgetting rules as below.

Forgetting Rule 1 In the end of each step during the local search phase, if v is uncovered by the candidate solution, $pc[v]$ will be increased by 1.

Forgetting Rule 2 In the end of each step after the local search phase, the *pc* of each vertex v is reduced to forget the earlier path cost decisions with the formula $pc(v) = pc(v)^*\beta + 1$, where β is a parameter.

Based on this path cost heuristic, we also define another definition, i.e., path scoring, to select which vertex need to be added or removed based on the circumstance of candidate solution.

Definition 6 (*Path scoring*) Given an undirected graph $G = (V,E)$, $pc(v)$ associated with each vertex $v \in V$, and a candidate solution *CS*, the path scoring denoted by *ps*, is a function such that

$$ps(v_i) = \begin{cases} \sum_{u \in N[v_i] \cap inde[u]=0} pc(u), & \text{for each } v_i \notin CS,\ inde[v_i] = 0 \\ -\sum_{u \in N[v_i] \cap inde[u]=1} pc(u), & \text{for each } v_i \in CS \end{cases}$$

where $inde[v]$ is used to denote the number of the close neighborhood of a vertex v covered by the candidate solution *CS*.

The search process is driven by a general and unified (v_1,v_2)-swap $(v_1,v_2 \in V)$ operation combined with specific rules to explore three constrained neighborhoods. Given an candidate solution *CS*, (v_1,v_2)-swap exchanges v_1 in $V \backslash CS$ against v_2 in *CS*, where there are no edges that v_1 is not adjacent to any vertex of $CS/\{v_2\}$. In the construction phase, to add the most promising vertex into the candidate solution, we use $(v_1,0)$-swap to iteratively generate the candidate solution. Then, in the local search phase, when obtaining an IDS, we use $(0,v_2)$-swap to find a smaller size of IDS. Otherwise, in the local search phases, we use (v_1,v_2)-swap to switch a pair of vertices with the help of specific selection rule.

A tabu mechanism [32, 33] is also employed to prevent searching from short-term cycles in local search phase. Tabu mechanism characteristically introduces a tabu tenure to forbid recently visited solutions from being revisited. In our algorithm, we adopt the following general prohibition rule: a vertex that leaves the current candidate solution CS is forbidden to be moved back to CS for the next T iterations where we use *tabu_list* to denote tabu tenure, which means that we allow this vertex to be a selected vertex, if this vertex is not included in *tabu_list*.

Our selection rules pick each (v_1,v_2)-swap operation by combining path scoring based on path cost with tabu mechanism which is formally like this.

(v_1,v_2)-swap operation rule Select v_2 from *CS* with the greatest $ps(v_2)$ value by breaking ties randomly, remove this vertex v_2 from *CS*, and put this just-removed vertex v_2 into *tabu_list*. Then, select v_1 from *V*/*CS* with the greatest $ps(v_1)$ value and $v_1 \notin$ *tabu_list* by breaking ties randomly and add this vertex v_1 into *CS*, where v_1 is not adjacent to any vertex in *CS*.

4 GRASP with path cost and tabu mechanism for the MIDS

In this section, we design a GRASP with path cost and tabu mechanism for the MIDS (GRASP + PC for short). Algorithm 1 shows the pseudo-code for GRASP + PC.

Algorithm 1 GRASP+PC
1 Initialize a best solution *CS** and a candidate solution *CS*;
2 Initialize the *pc* of each vertex as 1;
3 while *running time is less than the given time limit* **do**
4 *CS*←ConstructionPhase();
5 *CS*←LocalSearchPhase(*CS*, ITER);
6 **if**(|*CS*|<|*CS**|) **then**
7 *CS**←*CS*;
8 **end if**
9 $pc(v)=pc(v)*\beta+1$, for each vertex $v \in V$;
10 end while

Lines 1 and 2 initialize the best-known candidate solution CS*, the candidate solution CS, and the path cost of each vertex. The loop from lines 3–10 corresponds to the GRASP with path cost and tabu mechanism. In each iteration, an initial solution is built by the greedy randomized construction phase in line 4. A locally optimal candidate solution CS with respect to $(v_1,0)$-swap and (v_1,v_2)-swap is computed by the local search phase in line 5. At the end of each step in the line 9, the path cost of each vertex will drop the previous value by formula $pc(v) = pc(v)*\beta + 1$. It is built to generate a candidate solution based on some useful vertex information in the next construction phase.

4.1 Construction phase

Algorithm 2 shows the pseudo-code of the construction phase algorithm, which is used to construct the initial candidate solution for GRASP + PC.

Algorithm 2 ConstructionPhase()
1 Initialize *CS*;
2 Initialize the path scoring *ps* of each vertex based on the path cost *pc* of each vertex;
3 while *CS is not a IDS* **do**
4 ps_{max}←MAX{$ps(v)$|$ps(v)>0, v \in V/CS$};
5 ps_{min}←MIN{$ps(v)$|$ps(v)>0, v \in V/CS$};
6 RCL←{v|$ps(v)>ps_{min}+\alpha(ps_{max}-ps_{min}), v \in V/CS$};
7 v_1←selecting one vertex v_1 from RCL randomly;
8 *CS*←updating *CS* by the $(v_1,0)$-swap operation;
9 update $ps(v)$, for each vertex $v \in N(v_1)$;
10 end while
11 return *CS*;

The candidate solution represented by the candidate set *CS* is initialized in line 1. The path cost *pc* as well as the path scoring *ps* of all vertices are computed in line 2 for all vertices. The loop in lines 3–10 adds one vertex at each time to the candidate solution *CS*, until *CS* is an IDS. The maximum $ps_{\max}$ and minimum $ps_{\min}$ path scoring values of the candidate vertices are calculated in lines 4 and 5, respectively. The restricted candidate list (RCL), formed by all candidate vertices whose path scoring values are less than or equal to $ps_{\min} + \alpha(ps_{\max} - ps_{\min})$, is built in line 6, where α is a parameter in the interval [0,1]. A vertex v_1 is selected at random from the RCL in line 7, and the just-selected vertex is added to the candidate solution *CS* in line 8 through the $(v_1,0)$-swap operation. Finally, in line 9, the path cost value of neighborhood of this just-added vertex would be updated by judging the situation of candidate solution *CS*.

4.2 Local search phase

The candidate solutions built with the randomized greedy algorithm are not guaranteed to be locally optimal, even with respect to a simple neighborhood structure, i.e., only adding one vertex into the candidate solution each time. Therefore, applying local search phase into such a solution would definitely result in an improved locally optimal solution. We give this local search phase for MIDS in the following.

Starting from the candidate solution generated by the construction phase, the local search phase explores the neighborhood of the candidate solution for obtaining a smaller size of solution. If no improved solutions are found, the local search phase will return the constriction candidate solution as a local minimum. Otherwise, if an improved solution is obtained, it will be the new best candidate solution, and this phase repeats itself.

Our local search phase makes fully use of two operations including $(0,v_2)$-swap and (v_1,v_2)-swap. Specially, we use the first operation $(0,v_2)$-swap in attempt to remove some redundant vertices from the candidate solution. The second operation is (v_1,v_2)-swap in which we try to swap a pair of vertices where we need to remove v_2 from the candidate solution and then add v_1 into this solution. The local search phase is illustrated by the pseudo-code in Algorithm 3.

Algorithm 3 LocalSearchPhase(CS, ITER)
1 Initialize *tabu_list*;
2 for(*step*=0;*step*<ITER;*step*++)
3 **if**(*CS is IDS*) **then**
4 v_2←selecting v_2 from *CS* with the greatest $ps(v_2)$ value, breaking ties randomly;
5 update $ps(v)$, for each vertex $v \in N(v_2)$;
6 adding v_2 into *tabu_list*;
7 *CS*←updating *CS* based on the $(0,v_2)$-swap operation rule;
8 **continue**;
9 **end if**
10 v_2←selecting v_2 from *CS* with the greatest $ps(v_2)$ value, breaking ties randomly;
11 update $ps(v)$, for each vertex $v \in N(v_2)$;
12 adding v_2 into *tabu_list*;
13 v_1←selecting v_1 from $V \backslash CS$ with the greatest $ps(v_1)$ value and $v_1 \notin$ *tabu_list*, breaking ties randomly;
14 update $ps(v)$, for each vertex $v \in N(v_1)$;
15 *CS*←updating *CS* based on the (v_1,v_2)-swap operation rule;
16 $pc(v)=pc(v)+1$, for $v \in V/CS$;
17 update $ps(v)$, for each vertex $v \in V$;
18 end for
19 return *CS*;

At first, *tabu_list* is to be initialized. The loop in lines 2–18 is repeated until the step number reaches a maximum iteration number ITER. If the new obtained candidate solution is an IDS in lines 3–9, then our algorithm turns to find a smaller size of IDS by $(0,v_2)$-swap operation rule, i.e., finding a vertex with the greatest path scoring and then removing this vertex from the candidate solution. A new candidate solution will be tested for next steps. Otherwise, our algorithm will attempt to remove one vertex v_1 in candidate solution with the biggest path scoring value and update the path scoring of neighborhood of this removed vertex if the candidate solution is infeasible. In this process, the just-removed vertex v_1 in line 12 is inserted in *tabu_list*. After that, in the line 13 we select a greatest path scoring vertex v_2 not included in candidate solution or *tabu_list* and update the path scoring value of neighborhood of this vertex v_2. In line 15, we use the (v_1,v_2)-swap operation rule to update the candidate solution. After increasing the path cost value of each vertex by one, the corresponding path scoring value of each vertex also should be updated. Finally, when the number of step reaches ITER, the local search phase will return the candidate solution.

5 Results

In this section, we carry out extensive experiments to evaluate the performance of our algorithm GRASP + PC for MIDS on one classical benchmark, i.e., DIMACS (61 instances) [34].

For we are the first to design a heuristic algorithm to solve this problem, we choose two MIDS solvers, i.e., CPLEX12.6 and LocalSolver5.5 for comparison. LocalSolver5.5 is a new-generation, hybrid mathematical programming solver, which contains the best of all optimization techniques: local search, constraint propagation and inference, linear and mixed integer programming, as well as nonlinear programming techniques, while CPLEX12.6 is used to solve large-scaled linear programming problems as a commercial integer programming solver. Both of them run with some default parameter settings for solving MIDS. We run CPLEX12.6 and LocalSolver5.5 long enough so that they can acquire good approximations of the optima. The stopping criterion for CPLEX12.6 is either the convergence of lower and upper bounds or a maximum time limit set as 3600 s (1 h), while the stopping criterion of LocalSolver5.5 is a unified maximum same time limit 3600 s. Instead of comparing our algorithm GRASP + PC with CPLEX12.6 and LocalSolver5.5, which finds very good solutions for these instances, our aim is to use the solution values generated by CPLEX12.6 and LocalSolver5.5 as an indication of the quality of solutions by GRASP + PC.

There are four important parameters to be set in GRASP + PC. In the local search phase, we set the inner step to 10^6. For the RCL parameter, GRASP + PC set $\alpha = 0.8$ for all instances. GRASP + PC employs a tabu mechanism where the tabu tenure is set to 4. In our proposed new path cost strategy, the forgetting parameter β is set to 0.2.

Our algorithm GRASP + PC is implemented in C++ and compiled by g++ 4.6.2 with the –O2 option. All the experiments by GRASP + PC are run on Ubuntu Linux, with 2.3 GHZ CPU and 8 GB memory. For each instance, GRASP + PC performs 10 independent runs with different seeds, terminating upon reaching a given time limit (200 s) each time.

For each instance, *MIN* is the minimum independent dominating set found, *AVG* is the average size of 10 runs, *SD* is the standard deviation based on 10 results, and *Rtime*

Table 1 Experimental results of GRASP + PC, CPLEX12.6, and LocalSolver5.5 on DIMACS benchmark (61 instances)

Instance	GRASP + PC				CPLEX12.6		LocalSolver5.5
	MIN	AVG	SD	Rtime	UB	LB	UB
brock200_2	**4**	4	0	0.06	**4**	4	**4**
brock200_4	**6**	6.60	0.49	1.14	**6**	6	**6**
brock400_2	**10**	10	0	0.32	**10**	5.20	11
brock400_4	**9**	9.80	0.40	5.39	10	5.05	11
brock800_2	**8**	8.30	0.46	15.70	9	2.88	9
brock800_4	**8**	8.30	0.46	21.80	9	2.88	9
C1000.9	**27**	28	0.45	6.65	29	10.12	30
C125.9	15	15	0	0.62	**14**	14	**14**
C2000.5	**7**	7	0	5.49	11	2	8
C2000.9	**33**	33.20	0.40	52.12	48	10.03	36
C250.9	**17**	17.80	0.40	3.25	18	13.22	18
C4000.5	**8**	8	0	17.28	N/A	N/A	N/A
C500.9	23	23	0	2.78	23	11.39	**22**
c-fat200-1.clq	**13**	13	0	<0.01	**13**	13	**13**
c-fat200-2.clq	**6**	6	0	<0.01	**6**	6	**6**
c-fat200-5.clq	**3**	3	0	<0.01	**3**	3	**3**
c-fat500-1.clq	**27**	27	0	0.14	**27**	27	**27**
c-fat500-2.clq	**14**	14	0	<0.01	**14**	14	**14**
c-fat500-5.clq	**6**	6	0	<0.01	**6**	6	**6**
DSJC1000.5	**6**	6	0	1.29	**6**	2.06	**6**
DSJC500.5	**5**	5	0	3.20	10	2	7
gen200_p0.9_44	**16**	16.70	0.64	1.12	**16**	16	**16**
gen200_p0.9_55	**16**	16	0	1.11	**16**	16	**16**
gen400_p0.9_55	**21**	21.20	0.40	2.50	22	12.68	22
gen400_p0.9_65	**21**	21.60	0.49	4.31	**21**	12.67	22
gen400_p0.9_75	**20**	21.50	0.67	6.14	21	12.87	22
hamming10-4	**12**	12.80	0.40	13.46	14	5.82	**12**
hamming6-2	**12**	13.60	0.80	0.01	**12**	12	**12**
hamming6-4	**2**	2	0	<0.01	**2**	2	**2**
hamming8-2	**32**	34.40	2.42	1.03	**32**	32	**32**
hamming8-4	**4**	4	0	<0.01	**4**	4	**4**
johnson16-2-4	**8**	8	0	<0.01	**8**	8	**8**
johnson32-2-4	**16**	16	0	<0.01	**16**	16	**16**
johnson8-2-4	**4**	4	0	<0.01	**4**	4	**4**
johnson8-4-4	**7**	7	0	<0.01	**7**	7	**7**
keller4	**5**	5	0	0.01	**5**	5	**5**
keller5	**9**	9.90	0.54	9.79	11	4.14	10
keller6	**18**	18.80	0.40	174.22	32	5.59	19
MANN_a27	**27**	27	0	<0.01	**27**	27	**27**
MANN_a45	**45**	45	0	<0.01	**45**	45	**45**
MANN_a81	**81**	81	0	<0.01	**81**	81	**81**
MANN_a9	**9**	9	0	<0.01	**9**	9	12
p_hat1500-1.clq	**13**	13.40	0.49	59.81	17	4.06	19
p_hat1500-2.clq	**7**	7	0	51.40	9	2	9
p_hat1500-3.clq	**3**	3.20	0.40	56.04	4	1.33	4
p_hat300-1.clq	**9**	9	0	0.38	**9**	9	10
p_hat300-2.clq	**5**	5.10	0.30	0.16	**5**	5	**5**
p_hat300-3.clq	**3**	3	0	<0.01	**3**	3	**3**

Table 1 continued

Instance	GRASP + PC				CPLEX12.6		LocalSolver5.5
	MIN	AVG	SD	Rtime	UB	LB	UB
p_hat700-1.clq	**11**	11	0	1.69	14	4.80	13
p_hat700-2.clq	**7**	7	0	0.36	**7**	2.66	8
p_hat700-3.clq	**3**	3	0	0.09	**3**	3	**3**
san1000	**4**	4	0	11.62	**4**	2.99	5
san200_0.7_1	7	7	0	0.01	**6**	6	7
san200_0.7_2	**6**	6	0	0.01	**6**	6	**6**
san200_0.9_1	16	16	0	0.09	**15**	15	16
san200_0.9_2	**16**	16.80	0.40	0.13	**16**	16	**16**
san200_0.9_3	**15**	16	0.45	0.09	**15**	15	**15**
san400_0.5_1	**4**	4	0	0.03	**4**	4	**4**
san400_0.7_1	**7**	7.10	0.30	2.22	8	4.46	8
san400_0.7_2	**7**	7	0	0.66	**7**	4.59	8
san400_0.7_3	**8**	8	0	0.08	**8**	4.50	9

Table 2 Experimental results of GRASP + PC, GRASP + NOFORGET, and GRASP on DIMACS benchmark

Instance	GRASP + PC				GRASP + NOFORGET				GRASP			
	MIN	AVG	SD	Rtime	MIN	AVG	SD	Rtime	MIN	AVG	SD	Rtime
brock200_4	**6**	**6.6**	0.49	1.14	**6**	6.7	0.3	2.12	7	7	0	0.01
C250.9	**17**	17.8	0.4	3.25	18	18	0	0.61	18	18.1	0.3	2.4
gen200_p0.9_44	**16**	**16.7**	0.64	1.12	**16**	17.2	0.6	0.71	18	18.2	0.4	1.27
gen400_p0.9_75	**20**	21.5	0.67	6.14	21	21.7	0.46	3.28	21	22.5	0.67	4.79
hamming8-2	**32**	**34.4**	2.42	1.03	32	35.3	2	0.93	40	42.6	1.02	4.01
p_hat700-1	**11**	11	0	**1.69**	**11**	11	0	3.85	**11**	11.2	0.4	5.73
san200_0.9_3	**15**	16	0.45	0.09	**15**	16.1	0.54	1.81	16	16.2	0.4	0.69
san400_0.7_1	**7**	**7.1**	0.3	2.22	7	7.3	0.46	4.09	**7**	7.9	0.3	0.46

is the running time when GRASP + PC gets the minimum independent dominating set. We also list the best upper bound (UB) and lower bound (LB) values generated by CPLEX12.6 and the best upper bound (UB) of LocalSolver5.5. Bold face indicates the best solution values from GRASP + PC, CPLEX12.6, or LocalSolver5.5. For instances where CPLEX12.6 and LocalSolver5.5 fail to find a solution within the time limit, the column would be marked as "N/A".

The results based on DIMACS benchmark are summarized in Table 1. GRASP + PC is essentially better than CPLEX12.6 and LocalSolver5.5. In particular, compared with CPLEX12.6, GRASP + PC improves the minimum independent dominating set of 19 instances, while CPLEX12.6 outperforms our algorithm with the better quality of minimum solution in 3 instances. GRASP + PC can find better solutions than LocalSolver5.5 in 26 instances. However, in only 2 instances, the solution found by LocalSolver5.5 is better than that of the GRASP + PC. In the 39 ones out of all instances, GRASP + PC consistently finds the independent dominating set of the same quality.

5.1 The effectiveness of path cost strategy

In this subsection, we will evaluate our new path cost strategy and a counterpart forgetting strategy. Therefore, we design another two algorithms: GRASP + NOFORGET which adopts the path cost strategy into the local search and construction phases but does not apply our forgetting strategy to select some key vertices in these phases, and GRASP which works without two strategies.

The results are summarized in Table 2. In particular, compared with GRASP + NOFORGET, GRASP + PC

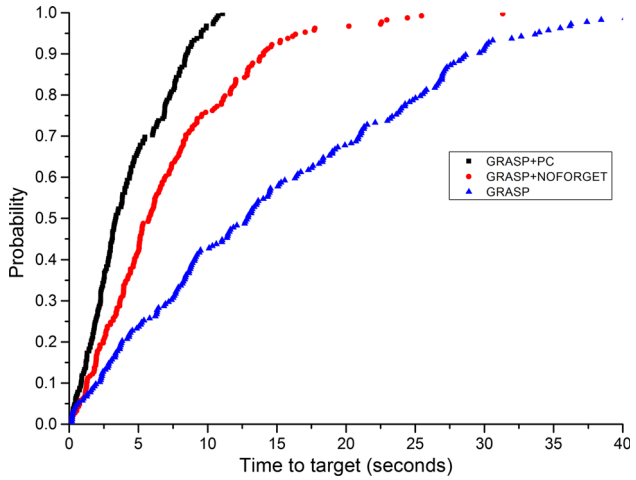

Fig. 1 Time-to-target plot comparing GRASP + PC with GRASP + NOFORGET and GRASP on instance p_hat700-1 and its target 11

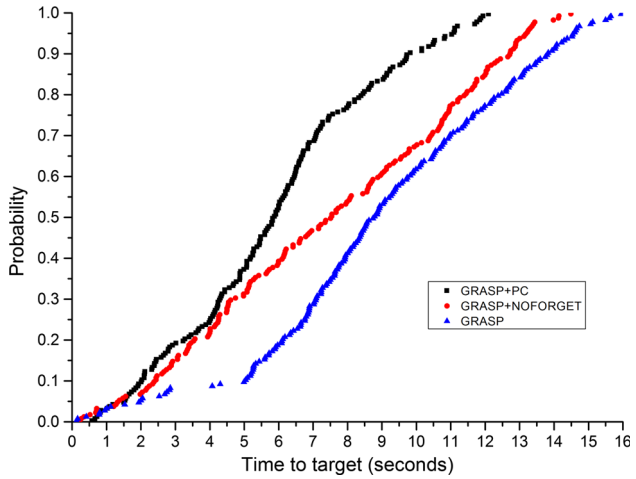

Fig. 2 Time-to-target plot comparing GRASP + PC with GRASP + NOFORGET and GRASP on instance san400_0.7_1 and its target 7

could improve the optimal values of two instances. When GRASP + NOFORGET and GRASP + PC find the same minimal value in two instances, GRASP + PC also gets the smaller value of average independent dominating set. For the rest of the instances where the two algorithms find solutions of the same quality, GRASP + PC consistently solves these instances faster than GRASP + NOFORGET. Compared with GRASP, GRASP + PC is also essentially better than GRASP.

To show the efficiency of our algorithm, as shown in Figs. 1 and 2, time-to-target plots [35] are used to compare GRASP + PC with GRASP + NOFORGET and GRASP on two instances, p_hat700-1 and its target 11, as well as san400_0.7_1 and its target 7, respectively. Two hundred independent runs of three algorithms are done for each of these problems. As shown in Fig. 1, the probability of obtaining a solution of the target value by GRASP + PC in at most 3.38 s is about 50 %, in at most 7.31 s is about 80 %, and in at most 8.54 s is about 90 %. However, the probability of obtaining a solution by GRASP + NOFORGET and GRASP of the target value in 3.86 s and 12.96 s is about 50 %, more than 7.76 s and 25.68 s about 80 %. Similarly, in Fig. 2, it is easy to observe that GRASP + PC performs much better than GRASP + NOFORGET and GRASP.

6 Conclusion

This paper presents the novel GRASP algorithm named GRASP + PC to solve MIDS. Few heuristics are available in the literature for this problem. The demand for good approximate algorithms for large-scale instances of this problem is well met, due to the fact that it is NP hard.

In this paper, the first heuristic is a novel path cost heuristic, which is motivated by observations on drawbacks of the original GRASP for MIDS. Based on this heuristic, we redesign the path scoring of each vertex to judge which vertex could be selected from the vertex set. Moreover, to diversify the candidate solution, we propose two forgetting rules to update the path cost value of each vertex in the construction and local search phases, respectively. To further deal with the classical cycling problem, we incorporate the tabu mechanism into the local search phase. Experiments on standard benchmarks show that GRASP + PC outperforms two current great solvers, CPLEX12.6 and LocalSolver5.5, in respect of MWCP. We also carry out extensive experiments to analyze the effectiveness of the path cost heuristic and corresponding forgetting rules.

This work takes the first step toward the heuristic algorithm for MIDS on classical graphs. In the future, we would like to design more efficient heuristic algorithms in the aspect of MIDS and to exploit other properties of vertices to promote improvement on such algorithms.

Acknowledgments The authors of this paper express sincere gratitude to all the anonymous reviewers for their hard work. This work was supported in part by NSFC under Grant Nos. (61272208, 61370156, 61403076, 61403077, 61402196), Program for New Century Excellent Talents in University (NCET-13-0724), and Jilin Province Science and Technology Development Plan under Grant Nos. (20140520067JH).

References

1. Erciyes K, Dagdeviren O, Cokuslu D et al (2007) Graph theoretic clustering algorithms in mobile ad hoc networks and wireless sensor networks. Appl. Comput. Math 6(2):162–180

2. Chen Y, Liestman A, Liu J (2004) Clustering algorithms for ad hoc wireless networks. Ad Hoc and Sensor Networks 28:76
3. Lin CR, Gerla M (1997) Adaptive clustering for mobile wireless networks. Selected Areas in Communications, IEEE Journal on 15(7):1265–1275
4. Nocetti FG, Gonzalez JS, Stojmenovic I (2003) Connectivity based k-hop clustering in wireless networks. Telecommunication systems 22(1–4):205–220
5. Basagni S (1999) Distributed clustering for ad hoc networks. In: Proceedings of the 1999 international symposium on parallel architectures, algorithms and networks. IEEE Computer Society, pp 310–315
6. Stankovic JA (2008) Wireless sensor networks. Computer 10:92–95
7. Santos AC, Bendali F, Mailfert J et al (2009) Heuristics for designing energy-efficient wireless sensor network topologies. Journal of networks 4(6):436–444
8. Akyildiz IF, Kasimoglu IH (2004) Wireless sensor and actor networks: research challenges. Ad Hoc Netw 2(4):351–367
9. McLaughlan B, Akkaya K (2007) Coverage-based clustering of wireless sensor and actor networks. In: IEEE international conference on pervasive services. IEEE, pp 45–54
10. Michael RG, David SJ (1979) Computers and intractability: a guide to the theory of NP-completeness. WH Freeman, San Francisco
11. Halldórsson MM (1993) Approximating the minimum maximal independence number. Information Processing Letters 46(4): 169–172
12. Goddard W, Henning MA (2013) Independent domination in graphs: a survey and recent results. Discrete Mathematics 313(7): 839–854
13. Johnson DS, Yannakakis M, Papadimitriou CH (1988) On generating all maximal independent sets. Information Processing Letters 27(3):119–123
14. Moon JW, Moser L (1965) On cliques in graphs. Israel journal of Mathematics 3(1):23–28
15. Gaspers S, Liedloff M (2006) A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. Graph-theoretic concepts in computer science. Springer, Berlin, pp 78–89
16. Liu C, Song Y (2006) Exact algorithms for finding the minimum independent dominating set in graphs. Algorithms and computation. Springer, Berlin, pp 439–448
17. Bourgeois N, Della Croce F, Escoffier B et al (2013) Fast algorithms for min independent dominating set. Discrete Applied Mathematics 161(4):558–572
18. Cai SW, Su KL, Luo C et al (2013) Numvc: an efficient local search algorithm for minimum vertex cover. J Artif Intell Res 46:687–716
19. Huang P, Yin M (2014) An upper (lower) bound for max (min) CSP. Science China Information Sciences 57(7):1–9
20. Gao J, Wang J, Yin M (2015) Experimental analyses on phase transitions in compiling satisfiability problems. Science China Information Sciences 58(3):1–11
21. Li X, Yin M (2015) Modified Cuckoo search algorithm with self adaptive parameter method. Inf Sci 298:80–97
22. Wang YY, Ouyang DT, Zhang L et al (2015) A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity. Sci China Inf Sci. doi:10.1007/s11432-015-5377-8
23. Wang YY, Cai SW, Yin MH (2016) Two efficient local search algorithms for maximum weight clique problem. In: AAAI
24. Li X, Zhang J, Yin M (2014) Animal migration optimization: an optimization algorithm inspired by animal migration behavior. Neural Comput Appl 24(7–8):1867–1877
25. Li X, Wang J, Yin M (2014) Enhancing the performance of cuckoo search algorithm using orthogonal learning method. Neural Comput Appl 24(6):1233–1247
26. Li X, Yin M (2014) Self adaptive constrained Artificial Bee Colony for constrained numerical optimization. Neural Comput Appl 24(3–4):723–734
27. Li R, Hu S, Wang Y, Yin M A local search algorithm with tabu strategy and perturbation mechanism for generalized vertex cover problem. Neural Comput Appl. doi:10.1007/s00521-015-2172-9
28. Festa P, Resende MGC (2002) GRASP: an annotated bibliography. Essays and surveys in metaheuristics. Springer, Berlin, pp 325–367
29. Festa P, Resende MGC (2009) An annotated bibliography of GRASP–Part I: algorithms. International Transactions in Operational Research 16(1):1–24
30. Festa P, Resende MGC (2009) An annotated bibliography of GRASP–Part II: applications. International Transactions in Operational Research 16(2):131–172
31. Glover F (1997) Tabu search and adaptive memory programming—advances, applications and challenges. Interfaces in computer science and operations research. Springer, Berlin, pp 1–75
32. Glover F (1989) Tabu search-part I. ORSA Journal on computing 1(3):190–206
33. Glover F (1990) Tabu search—part II. ORSA Journal on computing 2(1):4–32
34. Johnson DS, Trick MA (1993) Cliques, coloring, and satisfiability: second DIMACS implementation challenge, vol 26. American Mathematical Society, Providence
35. Aiex RM, Resende MGC, Ribeiro CC (2007) TTT plots: a perl program to create time-to-target plots. Optimization Letters 1:355–366