mathematics

MDPI

Article

A Novel Hybrid Algorithm for Minimum Total Dominating Set Problem

Fuyu Yuan, Chenxi Li, Xin Gao, Minghao Yin * and Yiyuan Wang *

School of Computer Science and Information Technology, Northeast Normal University, Changchun 130000, China; yuanfuyu@aliyun.com (F.Y.); icx935@nenu.edu.cn (C.L.); gaolzzxin@gmail.com (X.G.)

* Correspondence: ymh@nenu.edu.cn (M.Y.); yiyuanwangjlu@126.com (Y.W.)

Received: 14 January 2019; Accepted: 24 February 2019; Published: 27 February 2019

Abstract: The minimum total dominating set (MTDS) problem is a variant of the classical dominating set problem. In this paper, we propose a hybrid evolutionary algorithm, which combines local search and genetic algorithm to solve MTDS. Firstly, a novel scoring heuristic is implemented to increase the searching effectiveness and thus get better solutions. Specially, a population including several initial solutions is created first to make the algorithm search more regions and then the local search phase further improves the initial solutions by swapping vertices effectively. Secondly, the repair-based crossover operation creates new solutions to make the algorithm search more feasible regions. Experiments on the classical benchmark DIMACS are carried out to test the performance of the proposed algorithm, and the experimental results show that our algorithm performs much better than its competitor on all instances.

Keywords: minimum total dominating set; evolutionary algorithm; genetic algorithm; local search

1. Introduction

Given an undirected graph $G = (V, E)$, a dominating set (DS) is a subset of vertices $S \in V$ that each vertex in $V \backslash S$ is adjacent to at least one vertex in S. For each vertex $v \in V$, vertex v must have a neigbor in S, and this dominating set is called a total dominating set (TDS). We can easily conclude that TDS is a typical variant of DS. The minimum total dominating set (MTDS) problem aims to identify the minimum size of TDS in a given graph. MTDS has many applications in various fields, such as sensor and ad hoc communications and networks as well as gateway placement problems [1–3].

MTDS is proven to be NP-hard (non-deterministic polynomial) [4], which means unless P = NP, there is no polynomial time to solve this problem. At present, Zhu proposed a novel one-stage analysis for greedy algorithms [5] with approximation ratio $ln(\delta - 0.5) + 1.5$ where δ is the maximum degree of the given graph. This algorithm also used a super-modular greedy potential function, which was a desirable property in mathematics. However, in real life and industrial production, the size of problems is always very large. When the size of problems is increased [6–9], the approximation algorithm will be invalid. Considering these circumstances, researchers often use heuristic algorithms [10–13] to deal with these problems. Although the heuristic algorithms cannot guarantee the optimality of the solution they obtain, they can find high-quality solutions effectively within a reasonable time. Thus, in this paper we propose a hybrid evolution method combining local search and genetic algorithm (HELG) to solve MTDS.

Evolutionary algorithms include genetic algorithm, genetic programming, evolution strategies and evolution programming, etc. Among them, genetic algorithm as a classical method is the most widely used. Genetic algorithm is a computational model to simulate the natural selection and genetic mechanism of Darwin's biological evolution theory. It is a method to search the optimal solution by simulating the natural evolution process. Genetic algorithm begins with a population representing

the potential solution set of the problem, and a population consists of a certain number of individuals encoded by genes. After the first generation of the population, according to the principle of survival of the fittest, the evolution of generations produces more and better approximate solutions. In each generation, the selection is based on the fitness of the individual in the problem domain. Individuals, by means of genetic operators of natural genetics, perform crossovers and mutations to produce populations representing new solution sets.

Recently, evolutionary algorithms play an important role in solving optimization problems. It is common to adjust evolutionary algorithm to solve problems by adding a different problem-related mechanism. One possible improvement is the hybrid of evolutionary method and local search algorithm. Przewozniczek et al. investigated the pros and cons of hybridization on the baseof a hard practical and up-to-date problem and then proposed an effective optimization method for solving the routing and spectrum allocation of multicast flows problem in elastic optical networks [14]. Połap et al. proposed three proposition to increase the efficiency of classical meta-heuristic methods [15]. In this paper, the proposed algorithm takes advantage of local search framework as well as genetic algorithm.

Firstly, the algorithm creates a population including several individuals as initial solutions in our algorithm. Then, for each initial solution, we prove its solution via the local search. After local search, a repair-based crossover operation is proposed to improve the searchability of the algorithm. The algorithm randomly selects two solutions in the population and randomly exchanges several vertices of them. After crossover, if the obtained solutions are infeasible, the algorithm will repair them. This operation enables the algorithm to search larger areas, resulting in obtaining more feaisble solutions. In addition, we use a scoring function to help the method choose vertices more effective. In detail, each vertex is assigned to a cost value, and then we calculate the scoring value of every vertex by the cost value. The scoring function is used to measure the benefits of the state changing of a vertex. Whenever the algorithm swaps a pair of vertices, we should try to increase the benefits of the candidate solution and reduce the loss. This scoring value makes our algorithm efficient. When the original HELG fails to find improved solutions, this heuristic can make the algorithm escape from the local optimal.

Based on the above strategies, we design a hybrid evolutionary algorithm HELG for MTDS. Since we are the first to solve MTDS with a heuristic algorithm, in order to evaluate the efficiency of HELG, we carry out some experiments to compare HELG with a greedy algorithm and a classical metaheuristics algorithm, the ant colony optimization (ACO) algorithm [16,17] for MTDS. The experimental results show that on most instances our algorithm performs much better than the greedy algorithm.

The remaining sections are arranged as follows: in Section 2, we give some necessary notations and definitions. In Section 3, we introduce the novel scoring heuristic. The evolution algorithm HELG for MTDS is described in Section 4. Section 5 gives the experimental evaluations and the experimental results analysis. Finally, we summarize this paper and list future work.

2. Preliminaries

Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, each edge is a 2-element subset of V. For an edge $e = (u, v)$, vertices u and v are the endpoints of e, and u is adjacent to v. The distance between u and v means the number of edges from the shortest path of u to v and is defined by $dist(u, v)$. Then the ith level neighborhood of a vertex v is defined by $N_i(v) = \{u | dist(u, v) = i\}$. Specifically, $N(v) = N_1(v)$. The *degree* of a vertex v is $deg(v) = |N(v)|$.

A set D of V is a dominating set if each vertex not in D is adjacent to at least one vertex in D. Total dominating set is a variant of dominating set. The definition of total dominating set is as follows.

Definition 1. *(total dominating set) Given an undirected graph $G = (V, E)$, a total dominating set (TDS) is a subset D of V that every vertex of G is adjacent to some vertices in D, whether it is in D or not.*

The minimum total dominating set (MTDS) problem aims at identifying the TDS with the minimum size in a given graph.

3. Scoring Heuristic

In our algorithm, we need to maintain a total dominating set as a candidate solution CS during the construction and local search phases. It is important to decide which vertex should be added into or removed from CS. In this section, we will introduce an effective scoring heuristic that can be used to decide how to choose the vertices to be added and removed.

Given an undirected graph $G = (V, E)$, a cost value denoted by $\omega(v)$ is applied to each vertex $v \in V$, which will be maintained during the local search. For each vertex $v \in V$, $\omega(v)$ is initialized as 1 in the construction phase. In the local search phase, if CS uncovers the vertex v, the value of $\omega(v)$ will be increased by 1 such that these uncovered vertices will have more opportunities to be selected.

Based on the cost value described above, the vertex scoring method will be defined. We denote the scoring function as s, the scoring method is divided into two cases as follows:

- $v \notin CS$. The value of $s(v)$ is the total cost value of vertices which will become dominated by CS after adding vertex v into CS.
- $v \in CS$. The value of $s(v)$ is the opposite number of the total cost value of vertices which will become not dominated by CS after removing vertex v from CS.

The vertex scoring method is used to measure the benefits of changing the state of v. Obviously, if $v \in CS$, $s(v) \leq 0$. Otherwise, $s(v) \geq 0$. This method can decide how to choose the vertices to be added and removed.

4. Evolution Algorithm HELG for MTDS

In this section, we propose a hybrid evolutionary algorithm combining local search and genetic algorithm. The algorithm includes three important phases: generation of a population including n initial solutions, local search, and a repair-based crossover operation.

4.1. Population Initialization

We first generate a population including n initial solutions. We use a preprocessing method and the restricted candidate list (RCL) to initialize a population. In the process of preprocessing, based on the definition of MTDS, if the degree of a vertex is 1, its neighborhood will be added into CS and forbidden to be removed from CS during the search. A function *probid* is used to implement this process. For a vertex v, if $probid(v) = 1$, v will be forbidden to be removed from CS in the subsequent search. RCL contains the vertices with good benefits. The algorithm chooses vertices from RCL randomly to construct an initial candidate solution.

The pseudo code of construction phase is shown in Algorithm 1.

At first, the index k and population Pop are initialized (line 1). Then the n individuals are created (lines 2–18). For each individual, the scoring function s of each vertex, the *probid* value of each vertex and the candidate solution CS are initialized (lines 3–5). Then, the algorithm starts the preprocessing process (lines 6–9). If the degree of a vertex corresponds to 1, the algorithm adds its neighborhood into CS. Then the *probid* value of them will be assigned to 1 which means that they will be forbidden to be removed from CS in the following phase. The algorithm then enters a loop to add vertices into CS until it becomes a TDS (lines 10–16). The maximum value s_{max} and the minimum value s_{min} of s are calculated in lines 11 and 12. The vertices whose scoring values are not less than $s_{min} + \mu(s_{max} - s_{min})$ comprise the RCL (line 13). Here, μ is a parameter that belongs to [0, 1]. Then we choose a vertex u from RCL randomly and add it into CS (lines 14–15). The scoring values of u and the 1st and 2nd level neighborhood are updated in line 16. Then the just created individual is added into the Pop (line 17). Then, k is updated in line 18. In the end, we return the population Pop (line 19).

Algorithm 1: create_population (n)

Input: the population size n
Output: the initial population Pop

```
1  k ← 0, Pop ← {};
2  while k < n do
3      Initialize the scoring function s of each vertex based on the cost value ω;
4      Initialize the probid value of each vertex as 0;
5      CS ← {};
6      for each v ∈ V do
7          if deg(v) == 1 then
8              CS ← CS ∪ N(v);
9              probid(N(v)) ← 1;
10     while CS is not a TDS do
11         s_max ← MAX{s(v) > 0, v ∈ V/CS};
12         s_min ← MIN{s(v) > 0, v ∈ V/CS};
13         RCL ← {v|s(v) ≥ s_min + μ(s_max − s_min), v ∈ V/CS};
14         u ← choose a vertex u from RCL randomly;
15         CS ← CS ∪ {u};
16         update s(u), s(v1), and s(v2) for each vertex v1 ∈ N(u) and v2 ∈ N2(u);
17     Pop ← Pop ∪ CS;
18     k++;
19 return Pop;
```

4.2. Local Search Phase

Several candidate solutions are built in *create_population* phase. The local search phase explores the neighborhood of the initial candidate solution to improve the solution quality and obtain a smaller one. If no better solution is found, the algorithm will return the current solution as a local optimum. Otherwise, the improved solution will be the new best candidate solution. This phase is executed iteratively.

The pseudo code of local search phase is shown in Algorithm 2.

At the beginning of the algorithm, the number of iterations k and the local optimal solution CS^* are initialized (line 1). Then the algorithm enters a loop until the maximum number of iterations *mstep* is reached (line 2). In the search process, once a better solution is found, the algorithm updates CS^* by CS and removes a vertex with the highest s from CS (lines 3–9). The *probid* value of the selected vertex is forbidden to be 1. The number of iterations k is set to 0 (line 4). The scoring values of the just removed vertex and its 1st and 2nd level neighborhood are updated in line 8. Otherwise, the algorithm will remove a vertex v with the highest s and $probid(v) \neq 1$ from CS, breaking ties randomly (line 9). The corresponding scoring values are updated in line 11. Subsequently, the algorithm selects a vertex with the highest score and adds it to CS (lines 12–13). After that, the cost value of each undominated vertex v and the scoring function of each $v \in V$ are updated (lines 14–15). The step of iteration is increased by 1 (line 16). In the end, the algorithm returns CS^* as a local optimal solution.

Algorithm 2: LocalSearch (CS, $mstep$)

Input: an initial candidate solution CS, the maximum number of iterations $mstep$
Output: an improved candidate solution CS^*

Initialize $k \leftarrow 0$, $CS^* \leftarrow CS$;
while $k < mstep$ **do**
  if CS *is a TDS* **then**
    $k = 0$;
    $CS^* \leftarrow CS$;
    $v \leftarrow$ select v from CS with the highest $s(v)$, and $probid(v) \neq 1$, breaking ties randomly;
    $CS \leftarrow CS/\{v\}$;
    update $s(v)$, $s(u_1)$ and $s(u_2)$ for each $u_1 \in N(v)$ and $u_2 \in N_2(v)$;
  $v \leftarrow$ select v from CS with the highest $s(v)$ and $probid(v) \neq 1$, breaking ties randomly;
  $CS \leftarrow CS/\{v\}$;
  update $s(v)$, $s(u_1)$, and $s(u_2)$ for each $u_1 \in N(v)$ and $u_2 \in N_2(v)$;
  $v_1 \leftarrow$ randomly select undominated vertex v' and select v_1 from $N(v')$ with the highest $s(v_1)$, breaking ties randomly;
  $CS \leftarrow CS \cup \{v_1\}$;
  $\omega(v)$++ for each undominated vertex v;
  update $s(u)$ for each $u \in V$;
  k++;
return CS^*;

4.3. Repair-Based Crossover Operation

Genetic algorithms often use crossover to increase the diversity of the algorithm. The central role of biological evolution in nature is the recombination of biological genes (plus mutations). Similarly, the central role of genetic algorithms is the crossover operator of genetic algorithm. The so-called crossover refers to the operation of replacing the partial structure of two parent individuals to generate a new individual. Through crossover, the searchability of genetic algorithm has been greatly improved.

In this paper, we propose a repair-based crossover operation. After local search, our algorithm obtains an improved population. We choose two solutions from the population randomly, and then exchange the vertices in the two solutions with probability 0.5.

Because of the particularity of MTDS, the obtained solutions after crossover may be infeasible. So we should repair the infeasible solutions and make them become total dominating sets. After crossover, we check if the obtained solutions are total dominating sets. If a solution is infeasible, we add some reasonable vertices through population initialization phase until it is feasible. Then we perform a redundancy remove operation. For the solution obtained by crossover, we remove a vertex and check whether the solution is feasible. If it is feasible, the vertex will be removed, and otherwise it will be added back. The redundancy remove operation performs iteratively until every vertex has been checked.

The solution obtained by the repair-based crossover operation will replace the two old solutions into the population.

4.4. The Framework of HELG

In this paper, we propose a hybrid evolutionary algorithm HELG that combines local search and genetic algorithm. The algorithm first generates a population including n initial solutions, and then applies local search to improve each solution. The obtained n solutions will perform crossover to produce new solutions. This algorithm will perform iteratively until time limit is satisfied.

The framework of HELG is shown in Algorithm 3 and described as below.

Algorithm 3: HELG(G)

Input: an undirected graph G
Output: the best solution CS^* found

```
1  initialize CS*;
2  Pop = {S_1, ..., S_n} ← create_population(n);
3  while time limit is not satisfied do
4      for each CS ∈ Pop do
5          CS ← LocalSearch(CS, mstep);
6          if |CS| < |CS*| then
7              CS* ← CS;
8      Pop ← Crossover(Pop);
9      choose the best CS from Pop;
10     if |CS| < |CS*| then
11         CS* ← CS;
12 return CS*;
```

At first, the algorithm initializes the best solution CS^* and a population Pop (lines 1–2). Then the algorithm performs a loop (lines 3–11). For each solution CS of Pop, we use local search to improve the quality of solution and if CS is better than CS^*, CS^* is updated by CS (lines 4–7). For the obtained n solutions, we perform the repair-based crossover operation to generate new solutions (line 8). If the best solution CS among Pop is better than CS^*, CS^* is updated by CS (lines 9–11) When the time limit is satisfied, the best solution CS^* is returned (line 12).

5. Experiments

In this section, we carry out a series of experiments to evaluate the efficiency of our algorithm. The experiments are carried out on a classical benchmark DIMACS (the Center for Discrete Mathematics and Theoretical Computer Science) [18]. We select 61 instances in the DIMACS benchmark. The instances are from industry and generated by various models.

HELG is implemented in C++ and compiled by g++ with the -O3 option. We run the algorithm on a machine with Intel(R) Xeon(R) CPU E7-4830 @2.13Ghz and 4GB memory under Linux. For each instance, the HELG algorithm runs 30 times independently with different random seeds, until the time limit (100 s) is satisfied. HELG has three important parameters (i.e., n, μ and $mstep$). In the population creation phase, we set the RCL parameter $\mu = 0.1$ and $n = 10$. In the local search phase, we set the maximum number of iterations $mstep = 100$.

Since we are the first to solve MTDS with a heuristic algorithm, in order to evaluate the efficiency of HELG, a greedy algorithm is as our control method which uses the same scoring heuristic. At first, the candidate solution CS is empty. The greedy algorithm selects the vertex with the highest score value and adds it into CS every time. When CS becomes a TDS, the algorithm stops and returns CS as the optimal solution. Another comparison algorithm we use is a classical metaheuristics algorithm, the ant colony optimization (ACO) algorithm [16,17], which uses the same initialization procedure with our algorithm. We use this algorithm as a comparison algorithm to evaluate the effectiveness of our algorithm. For each instance, the ACO also runs 30 times independently with different random seeds, until the time limit (100 s) is satisfied.

For each instance, MIN is the minimum total dominating set found, AVG is the average size of 10 solutions, and $Time$ is the running time when the algorithm gets the minimum total dominating set. Because the greedy algorithm is only executed once for each instance, it has no average value and the time is less than 1. Better solutions and time are expressed in bold.

The experimental results are shown in Tables 1 and 2. Compared with the greedy algorithm, HELG can obtain better minimum solutions in 52 instances. In the remaining 9 instances, HELG gets the same minimum solutions with the greedy algorithm. Compared with ACO, HELG can obtain better

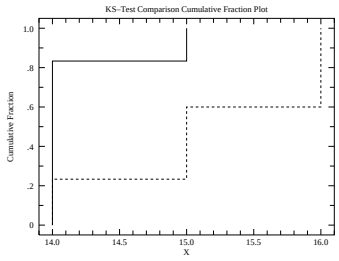
minimum solutions in 40 instances. In the remaining 21 instances, HELG gets the same minimum size with ACO. Among them, HELG gets better average values than ACO in 16 instances, and gets the same average value with ACO but performs faster in 5 instances. The DIMACS benchmark is divided into 10 groups. We choose 1 instance from every group. Every instance is run 30 times independently. The visualized comparisons of ACO and HELG can be seen by Kolmogorov-Smirnov test in Figure 1, which shows the distribution of the total dominating set values. From these, we can observe that HELG performs much better than ACO and greedy algorithm.

Table 1. Experimental results of greedy algorithm, ACO, and HELG on DIMACS I. The better minimum or average solution values are in bold.

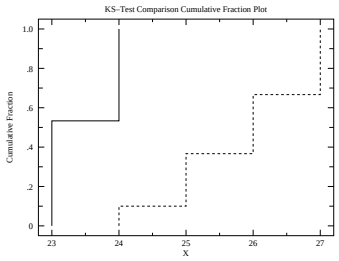
Instances	Vertices	Edges	Greedy		ACO			HELG		
			MIN	Time	MIN	AVG	Time	MIN	AVG	Time
brock200_2	200	10,024	7	<1	**6**	**6**	0.3	**6**	**6**	**0.21**
brock200_4	200	6811	10	<1	**9**	**9**	2.28	**9**	**9**	**1.93**
brock400_2	400	20,014	19	<1	16	16.3	6.28	**15**	**15**	5.39
brock400_4	400	20,035	19	<1	**14**	16.3	34.6	**14**	**15.1**	25.16
brock800_2	800	111,434	15	<1	13	14.2	15.2	**12**	**12.3**	10.5
brock800_4	800	111,957	18	<1	15	16.2	7.63	**13**	**13**	6.75
c-fat200-1.CLQ	200	1534	23	<1	22	22	48.62	**20**	**20.3**	43.93
c-fat200-2.CLQ	200	3235	12	<1	**10**	11	8.45	**10**	**10**	4.02
c-fat200-5.CLQ	200	8473	5	<1	5	5.3	13.51	**4**	**4**	2.46
c-fat500-1.CLQ	500	4459	53	<1	50	51.2	20.73	**47**	**48.1**	10.41
c-fat500-2.CLQ	500	9139	26	<1	24	25.9	79.62	**23**	**23.5**	74.99
c-fat500-5.CLQ	500	23,191	10	<1	**9**	**9**	11.12	**9**	**9**	**8.08**
C1000.9	1000	49,421	47	<1	45	45.6	68.45	**44**	**44.3**	64.97
C125.9	125	787	30	<1	21	22	45.2	**20**	**21.1**	38.16
C2000.5	2000	999,164	**9**	<1	10	10	46.7	**9**	**9.5**	43.37
C2000.9	2000	199,468	52	<1	51	52.4	17.89	**50**	**51.6**	11.64
C250.9	250	3141	33	<1	28	29	80.34	**27**	**28.2**	76.81
C4000.5	4000	3,997,732	**11**	<1	**11**	12.4	160.8	**11**	**11.6**	154.97
C500.9	500	12,418	39	<1	37	38.3	97.7	**34**	**36.6**	91.68
DSJC1000.5	1000	249,674	9	<1	**8**	9.1	11.6	**8**	**8.8**	7.38
DSJC500.5	500	62,126	9	<1	8	8	27.84	**7**	**7.4**	24.5
gen200_p0.9_44	200	1990	33	<1	**26**	27.2	77.1	**26**	**26.6**	**76.25**
gen200_p0.9_55	200	1990	31	<1	29	30.2	78.65	**25**	**26.2**	71.54
gen400_p0.9_55	400	7980	38	<1	35	35.5	53.2	**34**	**35**	51.65
gen400_p0.9_65	400	7980	38	<1	**33**	35	66.4	**33**	**34.3**	**58.49**
gen400_p0.9_75	400	7980	38	<1	35	36	27.5	**33**	**35.2**	20.4
hamming10-4	1024	89,600	23	<1	23	23	45.2	**21**	**22.6**	41.77
hamming6-2	64	192	19	<1	**15**	16.2	39.7	**15**	**15.6**	35.86
hamming6-4	64	1312	**3**	<1	**3**	**3**	0.01	**3**	**3**	**0.01**
hamming8-2	256	1024	69	<1	63	65	17.9	**62**	**64.4**	11.33
hamming8-4	256	11,776	9	<1	**6**	7.2	59.6	**6**	**6.5**	57.68

To further illustrate the efficiency contribution of our algorithm, we show the time-to-target plot [12,19] in Figure 2 to compare HELG with ACO on brock400_4 and its target 14. To obtain the plot, we performed 100 independent runs of each algorithm on brock400_4. The figure shows that the probabilities of finding a solution of the target value by HELG are approximately 30% and 70% in at most 13.78 and 33.43 s, respectively, whereas the probabilities of finding a solution of the target value by ACO are approximately 30% and 70% in at most 23.62 and 43.47 s respectively, considerably longer than HELG. From that, we can observe that the strategies we used in HELG are very effective.

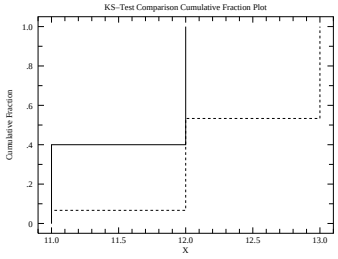
The experimental results show that our algorithm performs much better than the comparison algorithms. This proves that the genetic algorithm and local search in our algorithm are both very effective.



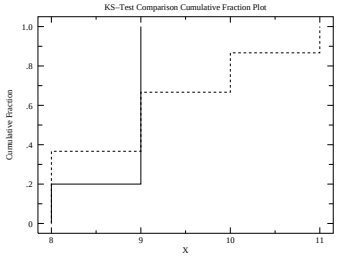
brock400_4



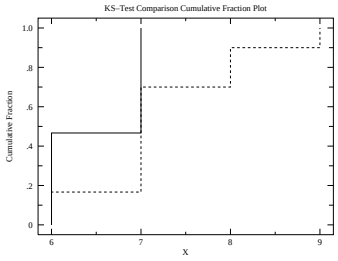
c-fat500-2.CLQ



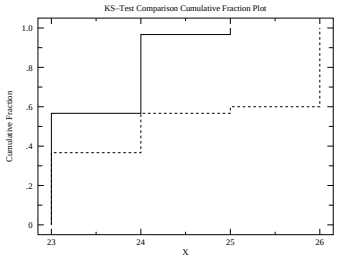
C4000.5



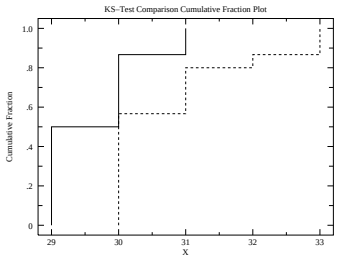
DSJC1000.5



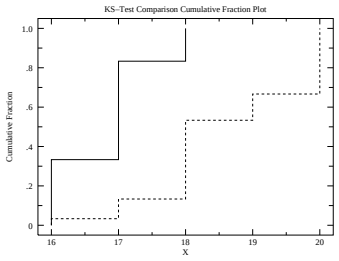
hamming8-4



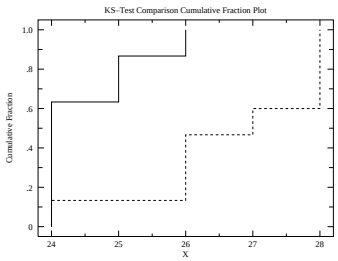
johnson32-2-4



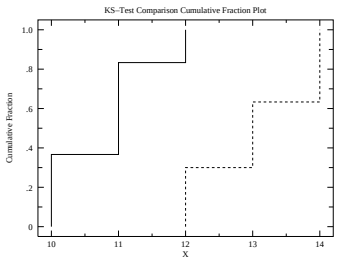
keller6



MANN_a9



p_hat1500-1.CLQ



san400_0.7_3

Figure 1. The total dominating set values obtained by · · · : ACO; —: HELG. Kolmogorov-Smirnov test can be applied to display the distribution of these values.

Table 2. Experimental results of greedy algorithm, ACO, and HELG on DIMACS II. The better minimum or average solution values are in bold.

Instances	Vertices	Edges	Greedy		ACO			HELG		
			MIN	Time	MIN	AVG	Time	MIN	AVG	Time
johnson16-2-4	120	1680	11	<1	**10**	11.2	32.6	**10**	**10.4**	30.72
johnson32-2-4	496	14,880	24	<1	**23**	24.5	34.8	**23**	**23.5**	33.38
johnson8-2-4	28	168	**5**	<1	**5**	5.8	0.01	**5**	**5**	0
johnson8-4-4	70	560	12	<1	8	8	16.43	**7**	**7.5**	9.81
keller4	171	5100	8	<1	8	8	10.45	**7**	**7.4**	4.52
keller5	776	74,710	16	<1	17	17	8.28	**15**	**16.5**	2.16
keller6	3361	1,026,582	30	<1	30	30.8	61.2	**29**	**29.6**	54.68
MANN_a27	378	702	113	<1	75	77.4	100.63	**67**	**79.5**	81.69
MANN_a45	1035	1980	**90**	<1	**90**	**90**	3.25	**90**	**90**	**0.03**
MANN_a81	3321	6480	**162**	<1	**162**	163	1.28	**162**	**162**	0.17
MANN_a9	45	72	20	<1	**16**	18.6	40.5	**16**	**16.8**	34.05
p_hat1500-1.CLQ	1500	839,327	26	<1	**24**	26.7	26.5	**24**	**24.5**	18.01
p_hat1500-2.CLQ	1500	555,290	11	<1	13	14.1	98.6	**8**	**9.2**	87.62
p_hat1500-3.CLQ	1500	277,006	5	<1	5	6.1	63.2	**4**	**4.9**	59.48
p_hat300-1.CLQ	300	33,917	16	<1	17	18.4	126.3	**14**	**15.2**	97.86
p_hat300-2.CLQ	300	22,922	**6**	<1	7	7	4.58	**6**	**6.5**	0.54
p_hat300-3.CLQ	300	11,460	4	<1	4	5	35.48	**3**	**3.1**	31.86
p_hat700-1.CLQ	700	183,651	21	<1	21	21	74.69	**17**	**20**	64.01
p_hat700-2.CLQ	700	122,922	9	<1	11	13	54.31	**8**	**10.2**	48.35
p_hat700-3.CLQ	700	61,640	**4**	<1	**4**	5	6.43	**4**	**4.4**	**3.25**
san1000	1000	249,000	**6**	<1	7	8	8.45	**6**	**6.5**	4.73
san200_0.7_1	200	5970	9	<1	**8**	9	6.7	**8**	**8.8**	1.57
san200_0.7_2	200	5970	10	<1	9	9.4	75.12	**7**	**7.9**	63.8
san200_0.9_1	200	1990	29	<1	24	24	30.5	**22**	**23.9**	25.07
san200_0.9_2	200	1990	30	<1	27	27.8	86.5	**24**	**25.9**	77.4
san200_0.9_3	200	1990	31	<1	27	27.6	45.65	**25**	**26.6**	34.13
san400_0.5_1	400	39,900	5	<1	6	6	8.41	**4**	**5.4**	2.12
san400_0.7_1	400	23,940	13	<1	13	13	27.4	**9**	**10.5**	19.49
san400_0.7_2	400	23,940	11	<1	10	11	100.6	**9**	**10.7**	92.22
san400_0.7_3	400	23,940	14	<1	12	13.1	87.45	**10**	**10.8**	69.77

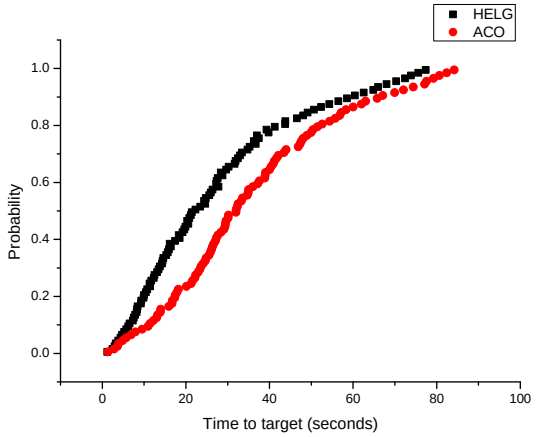


Figure 2. Time-to-target plot comparing HELG with ACO on instance brock400_4 and its target 14.

6. Summary and Future Work

This paper proposes a hybrid evolutionary algorithm combining local search and genetic algorithm to solve MTDS. A scoring heuristic is used to improve the efficiency of the algorithm. In the population initialization phase, we create a population including several initial solutions. In the local search phase, the algorithm improves the initial solutions by adding and removing operations. After that, we propose a repair-based crossover operation to increase the diversity of our algorithm. A series of experiments are carried out to evaluate the algorithm. The experimental results show that HELG performs well in solving MTDS.

In the future, we would like to design more efficient evolutionary algorithms to solve MTDS. We would like to relax this problem, such as the found MTDS is missing the adjacency of one vertex.

We will study evolutionary algorithms in more MDS-related problems, for example, the multi-objective MDS optimization problem.

Author Contributions: Software, X.G. and C.L.; Methodology, F.Y. and Y.W.; Writing–original draft preparation, F.Y. and C.L.; Writing—review and editing, C.L. and M.Y.

Funding: This research and the APC was supported by the Fundamental Research Funds for the Central Universities 2412018QD022 and Education Department of Jilin Province JJKH20190289KJ, NSFC (under grant nos. 61502464, 61503074, 61806050) and China National 973 Program 2014CB340301.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Subhadrabandhu, D.; Sarkar, S.; Anjum, F. Efficacy of misuse detection in ad hoc networks. In Proceedings of the 2004 First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, IEEE SECON 2004, Santa Clara, CA, USA, 4–7 October 2004; pp. 97–107.
2. Aoun, B.; Boutaba, R.; Iraqi, Y.; Kenward, G. Gateway Placement Optimization in Wireless Mesh Networks With QoS Constraints. *IEEE J. Sel. Areas Commun.* **2006**, *24*, 2127–2136. [CrossRef]
3. Chen, Y.P.; Liestman, A.L. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing, Lausanne, Switzerland, 9–11 June 2002; pp. 165–172.
4. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W.H. Freeman and Company: New York, NY, USA, 1990.
5. Zhu, J. Approximation for minimum total dominating set. In Proceedings of the International Conference on Interaction Sciences: Information Technology, Culture and Human, Seoul, Korea, 24–26 November 2009; pp. 119–124.
6. Cai, S.; Su, K.; Luo, C.; Sattar, A. NuMVC: An efficient local search algorithm for minimum vertex cover. *J. Artif. Intell. Res.* **2013**, *46*, 687–716. [CrossRef]
7. Ping, H.; Yin, M.H. An upper (lower) bound for Max (Min) CSP. *Sci. China (Inf. Sci.)* **2014**, *57*, 1–9.
8. Cai, S.; Lin, J.; Luo, C. Finding A Small Vertex Cover in Massive Sparse Graphs: Construct, Local Search, and Preprocess. *J. Artif. Intell. Res.* **2017**, *59*, 463–494. [CrossRef]
9. Wang, Y.; Cai, S.; Yin, M. Two Efficient Local Search Algorithms for Maximum Weight Clique Problem. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; pp. 805–811.
10. Wang, Y.; Cai, S.; Chen, J.; Yin, M. A Fast Local Search Algorithm for Minimum Weight Dominating Set Problem on Massive Graphs. In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), Stockholm, Sweden, 13–19 July 2018; pp. 1514–1522.
11. Wang, Y.; Cai, S.; Yin, M. Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *J. Artif. Intell. Res.* **2017**, *58*, 267–295. [CrossRef]
12. Wang, Y.; Li, C.; Sun, H; Yin, M. MLQCC: An improved local search algorithm for the set k covering problem. *Int. Trans. Oper. Res.* **2019**, *26*, 856–887. [CrossRef]
13. Wang, Y.; Cai, S.; Yin, M. New heuristic approaches for maximum balanced biclique problem. *Inf. Sci.* **2018**, *432*, 362–375. [CrossRef]
14. Przewozniczek, M.W.; Walkowiak, K.; Aibin, M. The evolutionary cost of baldwin effect in the routing and spectrum allocation problem in elastic optical networks. *Appl. Soft Comput.* **2017**, *52*, 843–862. [CrossRef]
15. Połap, D.; Kęsik, K.; Woźniak, M.; Damaševičius, R. Parallel Technique for the Metaheuristic Algorithms Using Devoted Local Search and Manipulating the Solutions Space. *Appl. Sci.* **2018**, *8*, 293. [CrossRef]
16. Romania, Q.S. Ant colony optimization applied to minimum weight dominating set problem. In Proceedings of the 12th WSEAS International Conference on Automatic Control, Modelling & Simulation, Catania, Italy, 29–31 May 2010.
17. Dorigo, M.; Stützle, T. Ant Colony Optimization. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999; pp. 1470–1477.

18. Johnson, D.S.; Trick, M.A. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11–13, 1993*; American Mathematical Soc.: Providence, RI, USA, 1996; Volume 26.
19. Aiex, R.M.; Resende, M.G.; Ribeiro, C.C. TTT plots: A perl program to create time-to-target plots. *Optim. Lett.* **2007**, *1*, 355–366. [CrossRef]