DOI: 10.3745/KIPSTA.2011.18A.2.069

Simulated Annealing 알고리즘을 이용한 최소 Dominating Set 문제의 효율성 증가에 대한 연구

정 태 의†

요 약

그래프 G의 최소 dominating set 문제는 G의 dominating set들 중 가장 작은 크기의 dominating set을 찾는 문제이며, NP-complete class에 속해 polynomial time안에 해결할 수 없는 문제로 잘 알려져 있다. 그러나, heuristic한 방법 혹은 approximation 방법을 이용해 특정한 분야에 적용이 가능하다. 본 논문에서는 세 개의 서로 다른 simulated annealing 알고리즘을 제시하여, 이들 알고리즘을 DIMACS에서 제시한 그래프들에 적용한 경우 효율성 증가가 이루어지는 것을 실험적으로 보이고자 한다.

키워드 : 최소 dominating set 문제, simulated annealing 알고리즘

Improving Efficiency of Minimum Dominating Set Problem using Simulated Annealing Algorithms

Tae Eui Jeong†

ABSTRACT

The minimum dominating set problem of a graph G is to find a smallest possible dominating set. The minimum dominating set problem is a well-known NP-complete problem such that it cannot be solved in polynomial time. Heuristic or approximation algorithm, however, will perform well in certain area of application. In this paper, we suggest three different simulated annealing algorithms and experimentally show better efficiency improvement by applying these algorithms to the graph instances developed by DIMACS.

Keywords : Minimum Dominating Set Problem, Simulated Annealing Algorithm

1. Introduction

Let $G = (V, E)$ be a simple undirected graph with vertex set $V = \{1, ..., n\}$ and edge set $E \subseteq V \times V$. Let n and m denote the number of vertices and edges, respectively. A *dominating set* of $G = (V, E)$ is a subset D of V such that every vertex not in D is adjacent to at least one vertex of D. A dominating set with minimum cardinality is called the *Minimum Dominating Set* (MDS). Minimum Dominating Set Problem (MDSP) of a graph G is the problem to find the minimum dominating set of G. The size of the MDS of G is called the *domination number* of G.

MDSP is a well-known NP-complete problem [2, 7]. The problem apparently cannot be solved in polynomial time. Because of such impracticality of developing an efficient algorithm for MDSP, it is much focused on the development of approximation [5, 6] or heuristic algorithms [3, 4] rather than the correct or optimal answer for some or all instances of the problem. In practice, it is possible that an approximation or heuristic algorithm will perform well experimentally, even if only for certain types of instances. MDSP, as an optimization problem, has numerous areas of application in the field of networks and communications. Abhay [3] suggested a greedy heuristic algorithm for MDSP, and Sanchis [4]

† 종신회원 : 서경대학교 컴퓨터과학과 교수
논문접수 : 2010년 3월 23일
수 정 일 : 1차 2011년 2월 8일
심사완료 : 2011년 2월 8일

recently suggested a randomized greedy algorithm for MDSP.

Since Kirkpatrick [8] first introduced the Simulated Annealing algorithms (SA) in 1983, the simulated techniques have been widely used for solving many combinatorial optimization problems. SA is very similar to the conventional iterative search algorithm with one major difference: SA allows permutations to escape from the local optimum in a controlled manner.

In this paper, we suggest three different simulated annealing algorithms called SA-random, SA-order, and SA-degree for MDSP. These algorithms differ in the sense that, while the first two algorithms are based on the randomness only, the third algorithm imposes suitable heuristic knowledge of MDSP. We also compare the performance of the algorithms by applying them to the graph instances developed by DIMACS.

The rest of the paper is organized as follows. In section 2, we first briefly introduce general structure of simulated annealing algorithm and explain the detail schemes of our three algorithms. Section 3 contains the experimental results. Finally, chapter 4 contains some conclusions.

2. Simulated annealing algorithms for the minimum dominating set

In this paper, we consider simple undirected graphs only. For notions and notations on simulated annealing algorithms and graph theory not explained here, please refer to [9] and [1], respectively. For a graph $G = (V, E)$, an ordering of V is a bijection $\beta:\{1, 2, .., n\} \leftrightarrow V$, where $n = |V|$. We denote $N(v)$ be the vertices adjacent to v in G.

For a graph $G = (V, E)$, a *solution* is a 0-1 vector $X = (x_1, x_2, ..., x_n)$ of length n, where each x_i is either 0 or 1. Let the subscript i of x_i represents the corresponding vertex of x_i in X and denote $D_X = \{ i \mid x_i = 1 \}$. Note that D_X is a subset of V. For example, if $n = 6$ and $X = (1, 0, 1, 0, 1, 1)$ then $D_X = \{1, 3, 5, 6\}$. Therefore, in this paper, we use the two notations X and D_X interchangeably.

If D_X is a dominating set of G, then we say that X is a *feasible* solution. Also let $f(X) = \sum_{i=1}^{n} x_i$ be the *fitness function* of a solution X. Then, MDSP is to find the solution X with minimum $f(X)$ among all possible feasible solutions.

SA is an iterative procedure that continuously updates one candidate solution to a new solution until a termination condition is met. Updating a solution is usually called move. Figure 2.1 shows the general structure of the proposed simulated annealing algorithms. It first generates an initial solution, which is a dominating set of a given graph, and continuously updates the current solution according to the move function. Let X be the current solution and Y be the solution generated by move function. Then, in line 10 of the following algorithm, if $f(X) > f(Y)$, then the function Accept_Solution() returns true. If $f(X) \leq f(Y)$, then Accept_Solution() returns true if $R < e^{\frac{-\Delta Gain}{T}}$, where R is a real random number in (0, 1). In all other cases, it returns false.

```
1   Begin
2       T = T0;
3       Tstop = Ts;
4       Current_Solution = Generation of a initial solution;
5       While Tstop > 0 do
6           accept = False;
7           For i = 1 to M do
8               New_Solution = Move(Current_Solution);
9               ΔGain = f(Current_Solution) - f(New_Solution);
10              If Accept_Solution(ΔGain, T) then
11                  Current_Solution = New_Solution;
12                  accept = True;
13              If accept then
14                  Tstop = Ts
15              Else
16                  Tstop = Tstop - 1
17              T = T * α
18  End
```

(Figure 1) General structure of the proposed SAs

Based on the algorithm shown in Fig. 1, our proposed three SAs differ only in line 4 and 8, i.e., the generation of initial solutions and move operations. The following three subsections contain the details of the three proposed SAs for MDSP.

2.1 SA-random

The mechanism of SA-random is based on the randomness. The initial solution X is generated randomly and forced to be a feasible solution by adding some vertices to X. The move operation is also performed in random fashion. We simply toggle the values of three randomly chosen bits. The following steps show the details of generating initial solution $X = (x_1, x_2, ..., x_n)$:

(Step 1) Randomly select some x_i's and set them to 1 and set all others to 0.

(Step 1) If X is not feasible then we randomly select xi which has value 0 and set $x_i = 1$. Repeat Step 1 until X becomes feasible.

For a current solution $X = (x_1, x_2, ..., x_n)$ the following two steps show the details of the move operation for SA-random:

(Step 1) Randomly choose three x_i's.

(Step 2) For each selected x_i apply $x_i + 1 \pmod 2$.

2.2 SA-order

Let S be a vector of length n whose elements are the permutation of integers in the range of $[1..n]$. Then S can be treated as a random ordering of the vertices of G. We visit the vertices of G according to S and build a dominating set in greedy manner. For the move operation we choose some subsequence of S and reverse the order of that subsequence. The following two algorithms *greedy-order* and *reverse* are used as initialization and move operation, respectively.

```
Algorithm: greedy-order(G, S)
begin
        D_X = ∅
        for i = 1 to n do
                v = S[i];
        if v ∈ G and (N(v) ∩ D_X = ∅) then
                D_X = D_X ∪ {v};

                delete v and N(v);
end
```

(Figure 2) Algorithm greedy-order

```
Algorithm: reverse(G, S, X)
begin
        Let p1 and p2 be the two unique integer values
        chosen randomly in the range of [1..n];
        Assume that p1 < p2 and Let S' = S;
        for i = 0 to p2 - p1 do
                S'[p1 + i] = S[p2 - i];
        S = S';
end
```

(Figure 3) Algorithm reverse

2.3 SA-degree

Unlike the SA-order, which chooses the vertices in random manner, we impose some criteria when choose the next vertex to be included in the minimum dominating set. If a vertex has larger degree then it covers more vertices than the vertice with small degree. Therefore it is quite reasonable to consider the vertice with larger degree first before considering the vertice with small degree when constructing a minimum dominating set. SA-degree uses this idea for constructing initial feasible solution and its detail is shown in Figure 2.4. Note that the algorithm greedy-maximal is also used for move operation for SA-degree for maximum perturbation of the solutions.

```
Algorithm: greedy-maximal(G)
begin
1       D_X = ∅;
2       while G ≠ ∅ do
3               let W be the set of vertices with
                maximum degree in G;
4               Randomly choose a vertex v from W
5               D_X = D_X ∪ {v}
6               delete v and N(v) from G
end
```

(Figure 4) Algorithm greedy-maximal

3. Experiments

Since there is no published benchmark graphs for MDSP, in order to measure the performance of our proposed three algorithms we run these algorithms on the 41 graph instances published by DIMACS[10], which originally developed for the problems of maximum independent set and minimum coloring. In all tests we used the following parameters: $T = 1000.0$, $\alpha = 0.98$, $T_s = 10$, and $M = 40$. Our experiments were run on a computer with a 2.33 GHz with 2 GB memory.

Table 1 shows the results of executing each algorithm ten times. For each algorithm the first three columns contains the result of best, worst, and average sizes of the dominating set for the corresponding graph among 10 executions of the algorithm. The fourth column shows the average execution time in seconds. For each algorithm *b*, *w*, and *ave* represent the best, worst, and average dominating size. The symbol *t* represents the average running time.

〈Table 1〉 Test Results

Graphs	n	m	Δ(G)	δ(G)	Three SA algorithms for MDSP											
					SA-random				SA-order				SA-degree			
					b	w	ave	t	b	w	ave	t	b	w	ave	t
Frb30-15-1	450	17827	122	42	15	20	17.6	7.9	14	17	16.1	1.1	12	12	12	4.6
Frb30-15-2	450	17874	116	45	18	24	19.9	7.9	14	17	16.0	1.1	12	12	12	4.1
Frb30-15-3	450	17809	122	49	17	21	18.7	7.9	15	17	16.3	1.1	12	12	12	4.2
Frb30-15-4	450	17831	110	48	16	23	19.5	7.9	15	17	16.1	1.1	11	11	11	4.1
Frb30-15-5	450	17794	128	46	18	23	19.6	7.9	16	18	16.6	1.1	12	12	12	4.6
Frb35-17-1	595	27856	132	50	19	26	22.2	14.0	18	21	19.6	1.7	14	14	14	8.5
Frb35-17-2	595	27847	134	53	18	27	23.0	13.2	18	20	18.9	1.7	14	14	14	8.2
Frb35-17-3	595	27931	165	45	20	25	22.9	13.7	18	19	18.8	1.7	14	14	14	8.3
Frb35-17-4	595	27842	150	34	21	25	23.4	13.9	19	21	19.8	1.7	14	14	14	9.5
Frb35-17-5	595	28143	134	44	22	26	23.7	14.0	18	20	18.9	1.7	14	14	14	8.2
Frb40-19-1	760	41314	178	56	25	31	28.8	22.1	22	24	23.1	2.3	16	16	16	15.3
Frb40-19-2	760	41263	171	57	24	33	28.6	22.1	21	24	22.2	2.4	16	17	16.1	18.1
Frb40-19-3	760	41095	159	57	25	29	27.3	22.2	20	23	22.1	2.3	16	16	16	17.3
Frb40-19-4	760	41605	164	67	25	30	28.2	22.3	20	23	21.8	2.3	16	16	16	15.8
Frb40-19-5	760	41619	174	68	25	32	28.1	22.3	21	22	21.7	2.3	15	15	15	17.9
Frb45-21-1	945	59186	188	68	29	38	32.1	34.4	23	26	24.9	3.2	18	18	18	26.5
Frb45-21-2	945	58624	191	74	27	35	30.9	33.9	24	26	25.4	3.2	18	18	18	26.9
Frb45-21-3	945	58245	205	72	27	39	32.6	34.2	22	26	24.7	3.2	17	17	17	27.0
Frb45-21-4	945	58549	212	69	28	36	31.9	34.6	24	27	25.3	3.1	18	18	18	26.7
Frb45-21-5	945	58579	180	70	27	32	30.0	34.1	22	27	25.0	3.2	18	18	18	27.9
Frb50-23-1	1150	80072	208	84	32	40	35.0	49.5	27	29	28.1	4.2	19	19	19	46.4
Frb50-23-2	1150	80851	227	75	31	47	37.1	49.8	27	30	28.1	4.2	19	20	19.6	52.3
Frb50-23-3	1150	81068	204	71	34	43	36.5	49.8	25	29	27.8	4.2	20	20	20	40.8
Frb50-23-4	1150	80258	208	78	32	42	35.8	49.4	27	30	27.9	4.2	19	19	19	45.5
Frb50-23-5	1150	80035	227	69	31	42	36.6	49.7	26	29	27.5	4.2	19	19	19	42.5
Frb53-24-1	1272	94227	232	86	35	45	39.0	60.2	28	31	29.9	4.8	21	21	21	50.1
Frb53-24-2	1272	94289	232	89	33	44	39.1	60.2	27	31	29.4	4.8	21	21	21	51.6
Frb53-24-3	1272	94127	234	88	38	46	40.6	60.3	28	33	29.6	4.9	20	20	20	49.3
Frb53-24-4	1272	94308	226	82	34	41	36.4	59.7	28	31	29.4	4.8	20	20	20	48.7
Frb53-24-5	1272	94227	206	72	33	42	36.9	60.2	28	30	29.1	4.8	21	21	21	51.3
Frb56-25-1	1400	109676	237	88	37	45	41.5	72.8	31	34	32.3	5.6	22	22	22	62.6
Frb56-25-2	1400	109401	237	72	34	45	39.6	72.3	30	33	31.7	5.6	22	22	22	62.7
Frb56-25-3	1400	109379	239	98	36	49	41.8	72.0	31	33	31.9	5.5	21	22	21.7	76.6
Frb56-25-4	1400	110038	241	92	37	43	40.2	72.9	29	33	31.4	5.7	22	22	22	63.9
Frb56-25-5	1400	109601	241	88	37	45	40.9	72.4	30	34	31.5	5.5	21	21	21	61.9
Frb59-26-1	1534	126555	277	94	40	50	43.5	86.3	32	36	33.4	6.3	22	22	22	84.8
Frb59-26-2	1534	126163	265	100	37	46	42.3	86.3	31	35	33.5	6.4	22	22	22	76.7
Frb59-26-3	1534	126082	258	78	38	49	43.2	87.1	32	34	33.2	6.7	23	23	23	76.3
Frb59-26-4	1534	127011	274	98	40	55	44.8	87.4	31	35	33.0	6.3	22	22	22	97.3
Frb59-26-5	1534	125982	238	81	38	53	46.2	86.3	32	34	32.7	6.4	23	23	23	86.1
Frb100-40	4000	572774	446	135	73	106	83.6	609.1	57	60	58.4	25.3	39	39	39	760.4
DSJC250.1	250	3218	38	13	22	26	23.9	2.1	21	28	24.4	0.7	16	16	16	1.8
DSJC500.1	500	12458	68	34	30	36	32.5	8.1	28	33	30.6	1.7	20	20	20	8.1
DSJC1000.1	1000	49629	127	68	37	46	41.0	34.1	34	39	37.1	4.1	23	23	23	39.5
DSJR500.1	500	3555	25	4	53	62	55.3	6.6	47	54	52.0	2.3	42	44	43.1	17.9

Table 1 shows that the performance of SA-degree is much better than those of the other two algorithms. This is due to the fact that unlike the other two algorithms, SA-degree does not totally depend on randomness. SA-degree uses the heuristic of larger degrees of the graphs. Comparing the best and worst cases of each algorithm, it can be easily seen that the performance of SA-degree are very steady. For the 29 test graphs only 4 graphs show the different best and worst size of dominating sets. These results show that, rather than based on total randomness, if we add some heuristic knowledge of the problem, then the performance of the simulated annealing algorithm can be greatly improved.

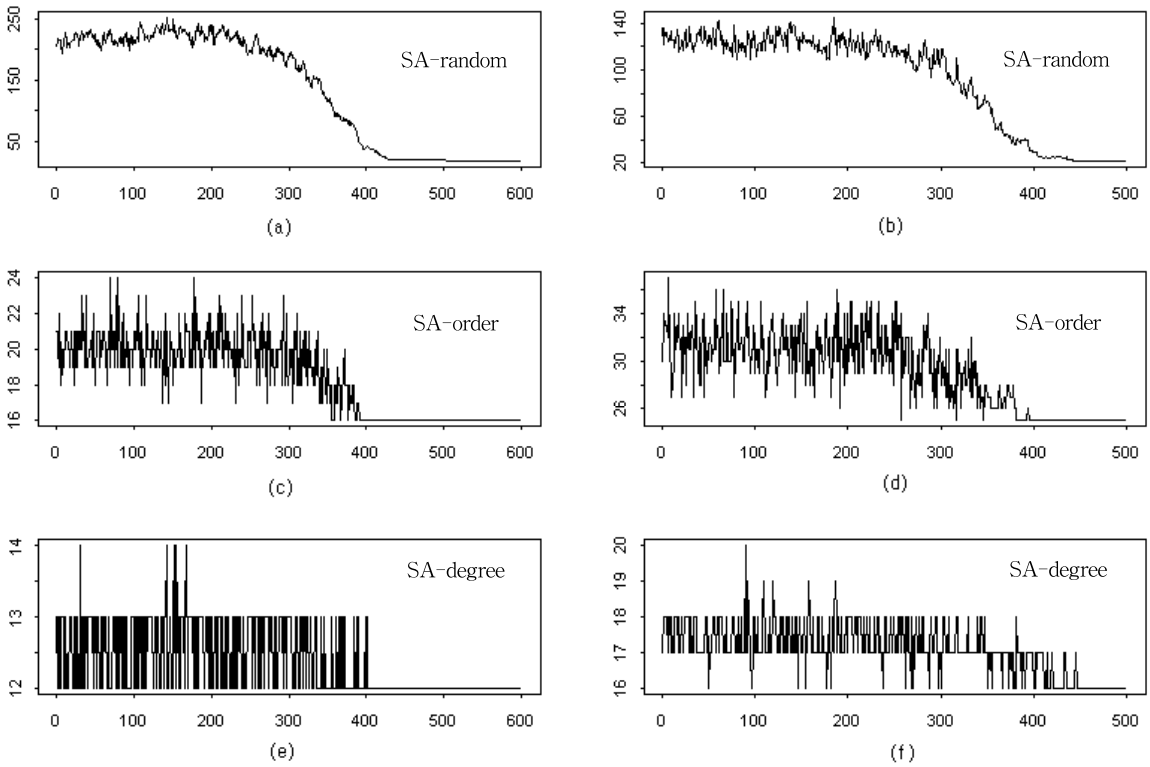
Figure 5 shows the convergence ratios of the three algorithms for the graphs Frb30-15-1 and DSJC250.1, respectively. For the graph Frb30-15-1, when the number of iterations closes to 400 the fitness values are start to converge. However, from the Fig. 5(e), it is easy to see that SA-degree shows steadier performance compare to the other two algorithms. Similar observations can be obtained from the graph DSJC250.1.

4. Conclusion

In this work, we showed that simulated annealing can be used to efficiently approximate the size of the minimum dominating set of graphs. For these purposes, we proposed three simulated annealing algorithms and measured the performance of these algorithms by applying them to the widely known graph instances. The results of the experiments clearly show that, by adding some suitable heuristic knowledge of the problem, it may improve the quality of the solutions when we search the solution space.

References

[1] West. B, “Introduction to Graph Theory”, Hall & Co.. 2000.
[2] Teresa W. Haynes, Stephen T. Hedetniemi, Peter J. Slater, “Fundamentals of Domination in Graph”, Maecel Dekker. 1998.
[3] Abhay, K. Parekh, “Analysis of a Greedy Heuristic for Finding Small Dominating Sets in Graphs”, Information Processing Letters, Vol.39, pp.237-240, 1991.
[4] L. A, “Experimental Analysis of Heuristic Algorithms for the Dominating Set Problem”, Algorithmica, Vol.33, pp.3-18, 2002.
[5] F. Kuhn. and R. Wattenhofer, “Constant-time distributed



(Figure 5) Graphs (a), (c), (e) and (b), (d), (f) show the convergence ratios of the three algorithms for the graphs Frb30-15-1 and DSJC250.1, respectively. *x*- and *y*-axis represent the number of iterations and fitness values, respectively

dominating set approximation", Distrib. Comput., Vol.17, pp. 303-310, 2005.

[6] V. Turau, "Linear Self-Stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler" Information Processing Letters, Vol. 103, pp.88-93, 2007.

[7] Garey, M. R. and Johnson, D. S. Computers and Intractability "A Guide to the Theory of NP-Completeness", W. H. Freeman & Co., 1979.

[8] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," Science, Vol.220, No.4598, pp. 671-680, 1983.

[9] P. J. M. Van-Laarhoven, E. Aarts, Simulated Annealing: Theory and Applications, Kluwer, Dordrecht, 1987.

[10] Johnson, D.S., Trick, M.A., eds.: Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge. Vol.26 of DIMACS Series. American Mathematical Society, 1996.

정 태 의

e-mail : tejeong@skuniv.ac.kr

1979년 고려대학교 전자공학과(학사)

1982년 미국 오하이오주립대 전기공학과 (석사)

1989년 미국 오클라호마대학 전산학과 (석사)

1994년 미국 오클라호마대학 전산학과(박사)

1983년 10월~1986년 6월 금성반도체연구소 컴퓨터부문 선임연구원

1986년 7월~1987년 8월 Engineering Manager, United Microtek, Inc.(San Jose, California)

1995년 3월~현 재 서경대학교 컴퓨터과학과 교수

관심분야 : Computational Complexity, Formal Languages, Graph Languages, 알고리즘, 이동통신