Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 156 (2008) 218–229
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Constructive generation of very hard 3-colorability instances

Kazunori Mizuno[a, b], Seiichi Nishihara[b]

[a] *Department of Computer Science, Takushoku University, Hachioji, Tokyo 193-0985, Japan*
[b] *Department of Computer Science, University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan*

Received 20 May 2004; received in revised form 6 December 2005; accepted 18 July 2006
Available online 18 April 2007

Abstract

Graph colorability (COL), is a typical constraint satisfaction problem to which phase transition phenomena (PTs), are important in the computational complexity of combinatorial search algorithms. PTs are significant and subtle because, in the PT region, extraordinarily hard problem instances are found, which may require exponential-order computational time to solve. To clarify PT mechanism, many studies have been undertaken to produce very hard instances, many of which were based on generate-and-test approaches. We propose a rather systematic or constructive algorithm that repeats the embedding of 4-critical graphs to arbitrarily generate large extraordinarily hard 3-colorability instances. We demonstrated experimentally that the computational cost to solve our generated instances is of an exponential order of the number of vertices by using a few actual coloring algorithms and constraint satisfaction algorithms.

Keywords: Graph coloring; Search; Phase transition; NP-complete; Hard problem; Heuristics; Constraint satisfaction

1. Introduction

Graph colorability (COL) as well as propositional satisfiability (SAT) is a typical constraint satisfaction problem, which has been studied from the viewpoint of computational complexity and search algorithms. Both of these combinatorial search problems are also very interesting as subjects of heuristics [2]. Many research reports discuss the computational complexity of COL [2–4,6,7,10,14,15,21–24]. Examples of possible candidates of order parameters that explain the mechanism making COLs extraordinarily hard include the 3-paths of Vlasie [22,23], the minimal unsolvable subproblem (MUS) of Mammen and Hogg [14], and the frozen developments of Culberson and Gent [4]. Since most of them are based on generate-and-test approaches, i.e., randomly generated instances, however, it may be difficult to inevitably obtain very hard instances.

Instead of traditional generate-and-test approaches, we propose a constructive approach producing graph 3-colorablity instances (3COLs) that are extraordinarily hard for actual coloring algorithms. Instances generated by our procedure (1) are themselves 4-critical, (2) contain no near-4-cliques (n4c's; 4-cliques with 1 edge removed) as subgraphs, and (3) are quasi-regular, i.e., have the degree of every vertex limited to 3, 4, or 5.

It is significant and useful to exploit such a method generating hard instances in giving some insights into the nature of hard instances and empirical evaluation of search algorithms.

E-mail addresses: mizuno@cs.takushoku-u.ac.jp, mizuno@npal.cs.tsukuba.ac.jp (K. Mizuno), nishihara@cs.tsukuba.ac.jp (S. Nishihara).

0166-218X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.07.015

In Section 2, after describing the definition of 3COLs and their phase transition phenomena, we explain characteristic structures for hard 3COL instances. In Section 3, we first propose seven graph structures for generating very hard instances, which we found by trial-and-error. We then give a generation algorithm using them. In Section 4, we show experimental results comparing the difficulty of our generated instances with that of randomly generated instances for some search algorithms.

2. Graph 3-colorability and 4-critical graphs

2.1. The graph 3-colorability problem

Let $G = (V, E)$ be a graph to be colored, where V and E correspond to the set of vertices and edges. Let $n = |V|$ and $m = |E|$. COL is an interesting example of a constraint satisfaction problem (CSP), where an edge $(i, j) \in E$ stands for the constraint that prohibits assigning the same color, or value, to vertices, or variables, i and j. 3COL, or COL with 3 colors available, has been in particular widely studied [2,4,6,11,14–16,22–25]. Phenomena similar to physical phase transitions are generally observed in CSPs, where search cost follows an easy–hard–easy pattern as a function of constraint density [3]. The region where median search cost becomes the most time-consuming lies very close to the cross-over point, at which half the instances are solvable and half unsolvable. This sort of phenomena is called primary PT.

An interesting region also exists at a slightly lower constraint density than that of primary PT, in which exceptionally hard instances (EHIs) [7,11,21] tend to occur, although most are solved easily. It is called secondary PT. For 3COLs with 100 vertices or less, primary PT and secondary PT are observed at $\gamma = 4.5$–4.6, and $\gamma = 3.3$–3.4, where γ is the average degree of the graph, i.e., $\gamma = 2m/n$, which corresponds to the doubled constraint density.

When 3COL instances are generated randomly, most EHIs in the secondary PT region are unsolvable [11]. Mammen and Hogg [14] have shown that the smallest minimal unsolvable subproblem is very large in hard unsolvable CSP instances. Vlasie [22] reports that hard instances tend to maintain regularity in vertex degrees, i.e., variance of vertex degrees is as small as possible. Despite some clarification of EHI properties, we have yet to develop fully direct methods that invariably generate EHIs.

2.2. Generation of large 4-critical graphs

A hard non-3-colorable graph necessarily contains a large size of 4-critical subgraphs [4,14].

Definition 1. A graph, G, is 4-critical if G itself is non 3-colorable, i.e., its chromatic number is 4, but any proper subgraph, $G'(\subset G)$, is 3-colorable.

K_4, 4-clique, is the smallest non-3-colorable graph (Fig. 1(a)). K_4 is 4-critical because removing an arbitrary edge from it makes a 3-colorable graph, which we call an n4c (Fig. 1(b)) [4]. The n4c implicitly contains an interesting constraint, say constraint (x, w) called a frozen pair [4], that claims the colors for x and w must be the same. Let Fig. 2(a) be a part of a 4-critical graph, where the degree of vertex i, $\deg(i)$, is 3. Introduce an operation, embed_$K_4(i, j)$, where an n4c (Fig. 1(b)) which is the near graph of K_4 is added in place of edge (i, j) merging i and x and connecting j and w. After one embedding operation (Fig. 2(b)), $\deg(i)$ becomes 4 while three new vertices with degree 3 are added.

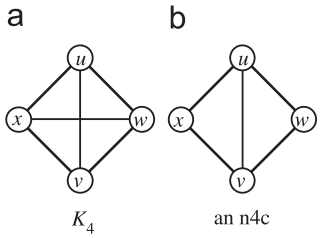

Fig. 1. Characteristic structures of 3COL.

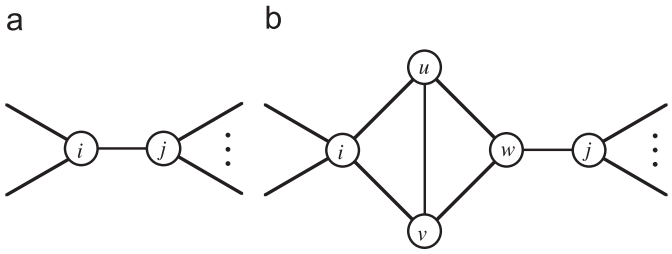

Fig. 2. Embedding operator embed_$K_4(i, j)$: (a) a 4-critical graph (part); (b) embedding completed.

This increments the number of vertices with degree 4 by 1, the number of vertices with degree 3 by 2, and the number of edges by 5. Note that 4-criticality is maintained because the constraint above, constraint (i, w), remains after embedding while u and v are not adjacent to other vertices.

Starting with K_4 as the initial graph, arbitrarily large 4-critical instances are constructed by repeating embed_$K_4(i, j)$ recursively to meet the many known conditions EHIs may have to satisfy: (1) constructed graphs are quasi-regular [22] and unsolvable [11], (2) any constructed graph itself is a minimal unsolvable subproblem [4,14], and (3) constraint densities of graphs are all in the secondary PT region [11]. In fact, the complexity of the instances, to which the backtracking algorithm with Brélaz heuristics is applied, is of an exponential order of the number of vertices [16].[1] However, the graphs which are generated by embed_$K_4(i, j)$ always leave at least 2 n4c's as subgraphs. Techniques similar to Smallk coloring program [19] can easily solve the instances because they lead to chains of collapse, i.e., reduction of the graph by merging two vertices in frozen pairs, showing immediately that the instance is non-3-colorable. Thus, large 4-critical subgraphs are necessary for hard 3COL instances, but being only large is not sufficient [4,16].

3. The constructive generation algorithm

3.1. New 4-critical graphs with no n4c's

To overcome the drawback of embed_$K_4(i, j)$ described in Section 2.2, it is significant to introduce a set of original n4c-free 4-critical graphs, which are independent of each other in the sense that no graph is a subgraph of any other, to generate hard instances [16–18]. Fig. 3 gives seven such graphs that we found by trial-and-error, in each of which is termed MUG$_{nt}$, where MUG stands for "minimal unsolvable graph", n means the number of vertices included, and t is used to identify the type if necessary. Detailed structures are listed in Table 1,where m is the number of edges and n_i denotes the number of vertices with degree i, i.e., $n = n_3 + n_4 + n_5$. All graphs are quasi-regular graphs, i.e., the degree of every vertex is 3, 4, or 5, in all of which graphs except for MUG12a and MUG12b have the degree of every vertex only limited to 3 or 4. Let us discuss the minimality of MUG$_{nt}$. For $n \leqslant 3$, there are no MUGs because all possible graphs are 3-colorable. MUG4, which is an alias of K_4, is thus the minimal MUG but contains n4c's as subgraphs. Exhaustively analyzing graph structures, we found that all MUGs generated for $5 \leqslant n \leqslant 8$, as well as MUG4, contain 1 or more n4c's. Thus, the minimal n4c-free MUG is MUG9 shown in Fig. 3(a).

In Section 2.2, we generate larger 4-critical graphs by repeatedly applying embed_$K_4(i, j)$ to the initial graph, K_4. In other words, this operation embeds the near graph of a MUG, i.e., the graph where one edge is removed from the MUG, to the MUG. Thus, we can naturally extend the embedding operation, embed_$K_4(i, j)$, to embed_MUG$_{nt}(i, j)$ by embedding the near graph of one of seven MUGs in Fig. 3. These operations are the same as *Hajós join construction* [9,12] except that both vertices to be merged should have the degree of 3.

Proposition 1. *When embed_MUG$_{nt}(i, j)$ is applied to a* 4*-critical graph, the result remains* 4*-critical.*

[1] Our experiments [16] showed that solvable instances derived by removing one edge arbitrarily from 4-critical graphs described above also requires exponential-order computational cost against the Brélaz algorithm.

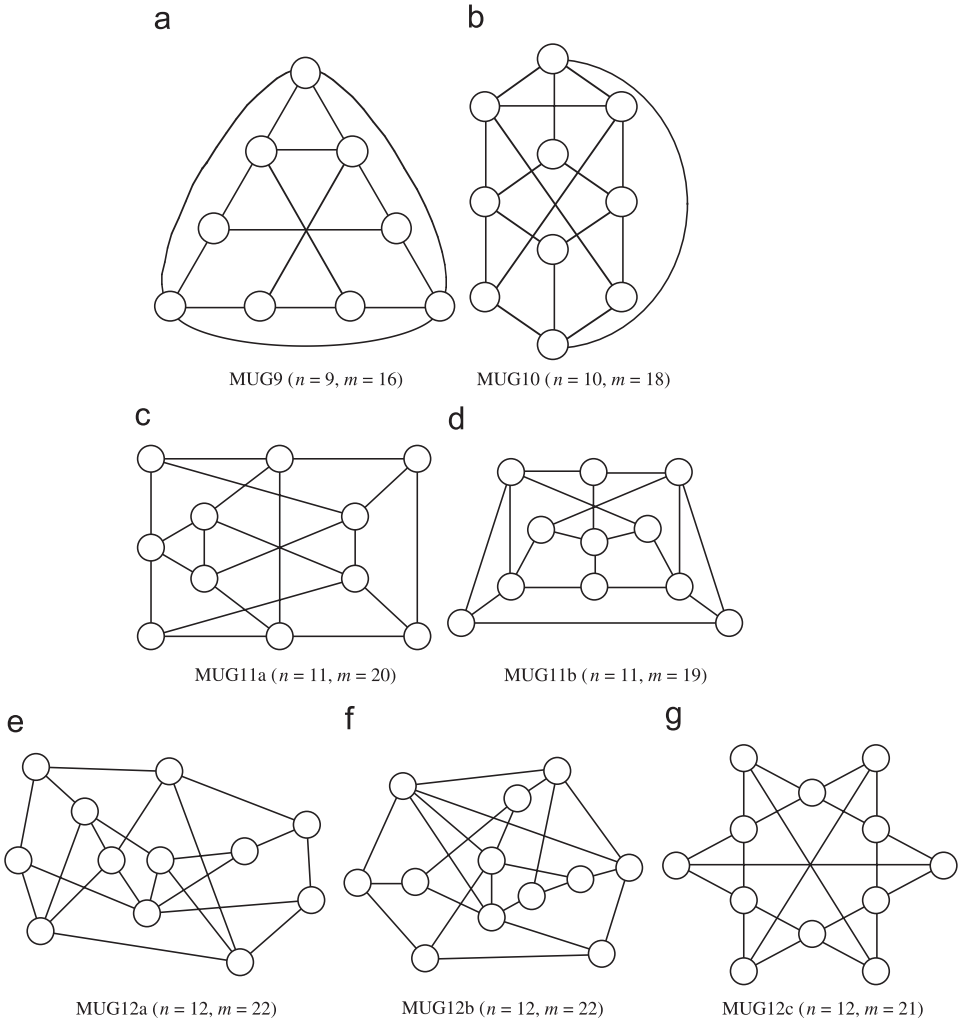

Fig. 3. 4-critical n4c-free graphs: (a) MUG9 ($n = 9, m = 16$); (b) MUG10 ($n = 10, m = 18$); (c) MUG11a ($n = 11, m = 20$); (d) MUG11b ($n = 11, m = 19$); (e) MUG12a ($n = 12, m = 22$); (f) MUG12b ($n = 12, m = 22$); (g) MUG12c ($n = 12, m = 21$).

Table 1
Numbers of components per n4c-free 4-critical graph

MUG_{nt}	n	m	$n = n_3 + n_4 + n_5$		
			n_3	n_4	n_5
MUG9	9	16	4	5	0
MUG10	10	18	4	6	0
MUG11a	11	20	4	7	0
MUG11b	11	19	6	5	0
MUG12a	12	22	5	6	1
MUG12b	12	22	7	2	3
MUG12c	12	21	6	6	0

```
procedure graph-generator(k)
begin
  input an initial graph G_init;                                   (1)
  G := G_init;
  for w := 1 to k do
    choose randomly an edge(i, j) ∈ E(G) where deg(i) ≤ 3;
    choose randomly MUG_nt, (nt =9, 10, 11a, 11b, 12a, 12b, or 12c); (2)
    embed_MUG_nt(i, j);
  end for;
end.
procedure embed_MUG_nt(i, j)
begin
  choose randomly an edge(x, y) ∈ E(MUG_nt) where deg(x) ≤ 3;
  remove the edge (i, j)
  remove the edge (x, y);
  add an edge (j, y);
  merge x with i;
end.
```

Fig. 4. The 3COL instance generator, where the argument k gives the number of times to apply the embedding operation.

Proposition 2. *When embed_MUG$_{nt}(i, j)$ is applied to a quasi-regular graph, quasi-regularity is maintained by embed_MUG$_{nt}(i, j)$, i.e., the degree of every vertex is limited to* 3, 4, *or* 5. *In particular, when nt is* 9, 10, 11*a*, 11*b*, *or* 12*c*, *the degree of every vertex is only limited to* 3 *or* 4.

Proposition 3. *Let the graph contain m edges and $n(=n_3+n_4+n_5)$ vertices, where n_i denotes the number of vertices with degree i. The number of vertices with degree* 3, 4, *and* 5 *increases by n_3-2, n_4+1, and n_5. The total number of vertices increases by $n-1$, and edges by $m-1$.*

3.2. The algorithm

Starting with a 4-critical graph as the initial graph, we construct arbitrarily large 4-critical graphs, i.e., including an arbitrary number of vertices, by repeating embedding. Fig. 4 gives the procedure "graph-generator (k)" which repeats embedding operations k times randomly. If one of the seven graphs in Fig. 3 is assigned initially to G_{init} at (1) in Fig. 4, and is chosen at (2) in Fig. 4, then the graph-generator can produce 4-critical and quasi-regular graphs with no n4c's as subgraphs. Fig. 5 gives an example of the 3COL instance generated after four embedding operations, i.e., graph-generator (4) is applied, where, starting with MUG10 as the initial graph, G_{init}, (Fig. 5(a)), embed_MUG9 is first applied to G_{init} (Fig. 5(b)) and the graph shown in Fig. 5(c) is finally derived as a result.

4. Experiments

4.1. Experimental results

We test the difficulty of 3COL instances generated by our procedure "graph-generator (k)" where all MUG$_{nt}$'s are used to generate n4c-free 4-critical graphs. For eight cases from $k=5$ to $k=12$, 300 3COL instances are generated at each case, i.e., a total of 2400 generated instances. These instances are applied to the backtracking algorithm with Brélaz heuristics [2] and Smallk coloring program [19], called Brélaz algorithm and Smallk shortly. In Brélaz algorithm, 1200 instances from $k=5$ to $k=8$ are used for testing. The number of assignments of colors to vertices is counted as search cost until the colorability is determined. In Smallk, all 2400 instances are used for testing and the number of "Search nodes" in it is measured as search cost. Both algorithms are implemented in C on a PC with 1 GHz of Pentium III and 512 Mbytes of RAM.

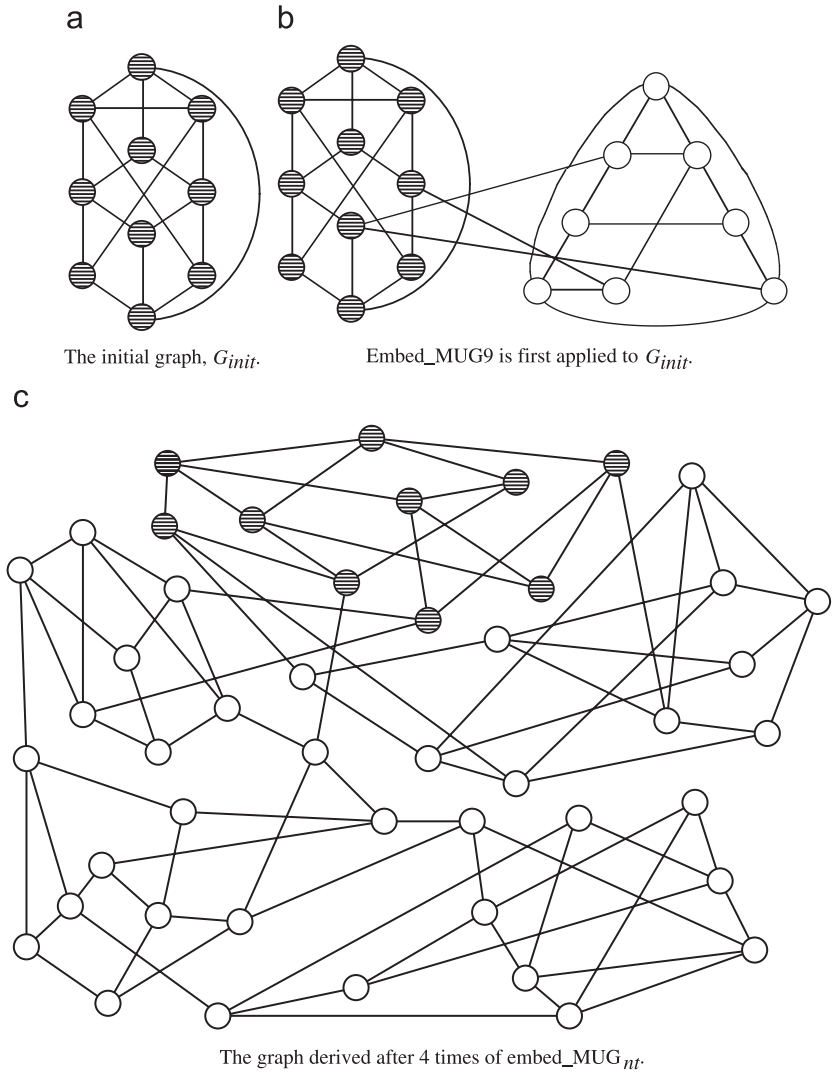

Fig. 5. Example of graph-generator-produced 3COL instance, whose subgraph composed of striped nodes shows the initial graph, G_{init}: (a) the initial graph, G_{init}; (b) embed_MUG9 is first applied to G_{init}; (c) the graph derived after 4 times of embed_MUG$_{nt}$.

Fig. 6 gives results for search costs and CPU time, where "Average" shows the variation in average search cost and CPU time for each k as a function of the average number of vertices for each k. Smallk is more sophisticated than Brélaz algorithm, but both search cost and CPU time clearly exhibit exponential growth.[2]

Fig. 7 gives results on "3-colorable" instances derived by removing one edge from 3COL instances used in Fig. 6. In Fig. 7(b), results for instances whose CPU time is less than 0.01 s are plotted at "1E-03". Although these results vary more widely than results in Fig. 6, "3-colorable" instances also appear to require exponential-order computational cost.

We tested the difficulty of other 3COL instances generated by repeatedly embedding only one type of the n4c-free 4-critical graph the same as the initial graph, G_{init}, i.e., generated instances are derived by setting a limit to the same MUG$_{nt}$ at both (1) and (2) in Fig. 4. Using each of the seven MUGs in Fig. 3, for three cases of $k = 6, 7$, and 8, 50 instances are generated at each case. All instances generated by using each of MUG9, MUG10, MUG11a, MUG11b, and MUG12c have the degree of every vertex limited to 3 or 4. Instances generated by using each of MUG12a and

[2] Compared to the results of Culberson and Gent [4], our instances seem to be much harder than their threshold graphs, although the complexity of their graphs exhibit exponential growth.

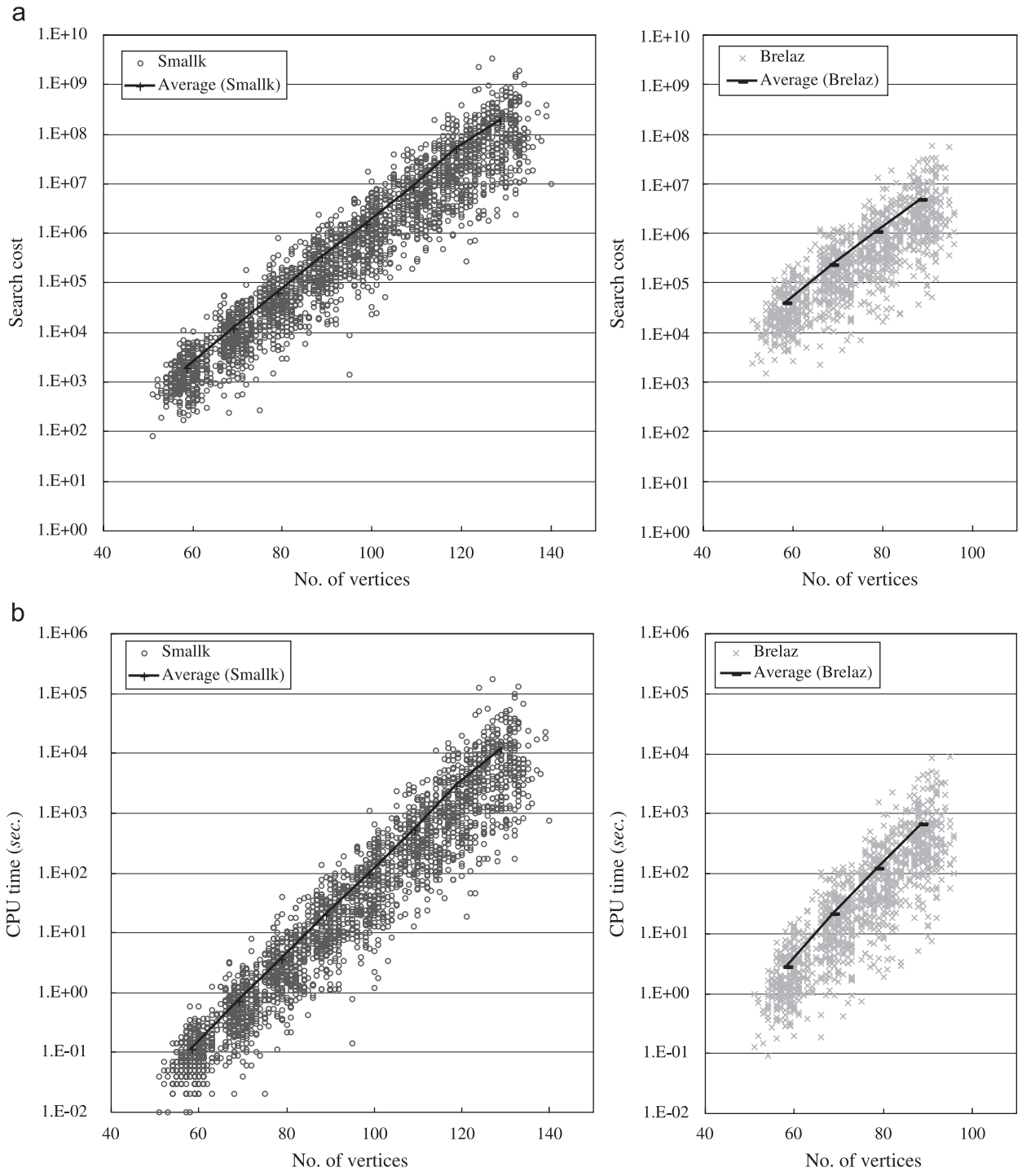

Fig. 6. Experimental results on 3COL instances generated by our procedure: (a) search cost; (b) CPU time.

MUG12b include some vertices whose degree is 5. Table 2 summarizes results, where Max, Mean, Median, and Min give statistics of search cost when each instance is solved using Smallk. As with results for Fig. 6(a), very hard instances can be generated. In particular, it is much harder for Smallk to solve instances generated by using only MUG11b compared

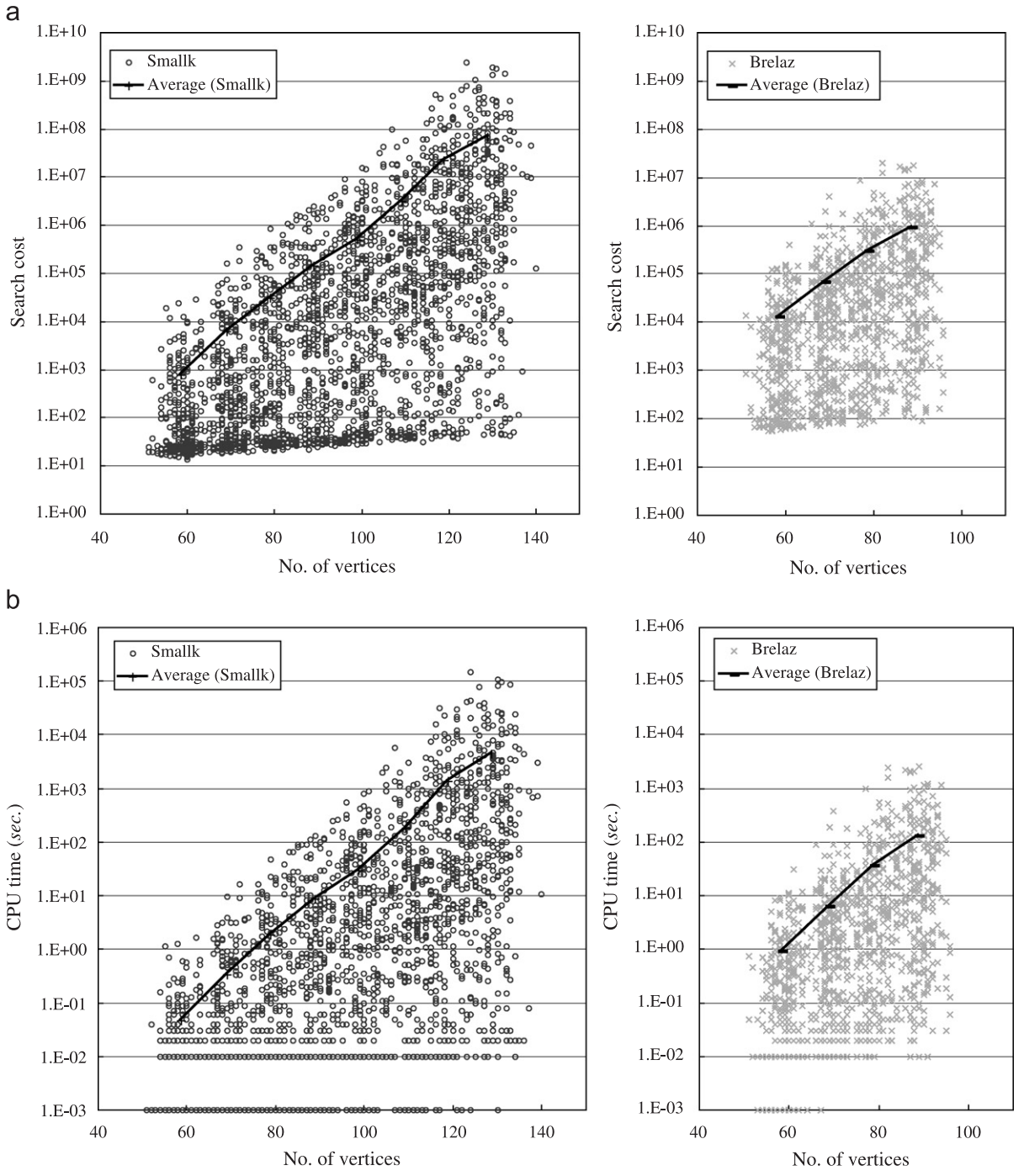

Fig. 7. Experimental results on "3-colorable instances": (a) search cost; (b) CPU time.

to other instances with the almost same number of vertices. MUG12b-based instances, in which variance of vertex degrees is larger than others, appear to be easiest to solve.

We also conduct experiments on randomly generated instances. For 33 cases from $\gamma = 3.0$ to $\gamma = 5.0$ at the intervals of 0.2 in $n = 100, 200,$ and 300, 10,000 instances are randomly generated at each case, i.e., a total of 3.3 million

Table 2
Experimental results on the instances generated by embedding only one type of the n4c-free MUG

k	MUG	n	Max	Mean	Median	Min
6	MUG9	57	2565	1035	919	223
	MUG10	64	18,143	9302	8155	3413
	MUG11a	71	30,039	12,233	10,884	3453
	MUG11b	71	601,986	282,030	279,329	74,974
	MUG12a	78	61,520	18,155	15,582	1713
	MUG12b	78	13,233	2635	2168	95
	MUG12c	78	7646	4161	4513	342
7	MUG9	65	11,423	3391	2646	919
	MUG10	73	119,034	42,402	37,411	14,836
	MUG11a	81	151,179	58,951	58,971	13,734
	MUG11b	81	5,269,203	2,537,279	2,315,360	734,768
	MUG12a	89	445,767	90,121	69,288	6384
	MUG12b	89	33,607	7111	4151	357
	MUG12c	89	179,858	22,842	17,752	4573
8	MUG9	73	43,767	13,423	11,103	2586
	MUG10	82	421,315	181,848	175,198	55,927
	MUG11a	91	899,150	279,396	220,238	67,427
	MUG11b	91	33,401,977	16,816,160	15,190,974	7,618,986
	MUG12a	100	1,983,599	556,850	441,649	74,526
	MUG12b	100	335,671	35,558	15,985	2252
	MUG12c	100	1,148,454	74,985	49,633	3061

k, MUG, and n show the number of times to apply the embedding operation, the graph used at both (1) and (2) in Fig. 4, and the number of vertices in generated instances, respectively, and max, mean, median, and min correspond to search cost in Smallk.

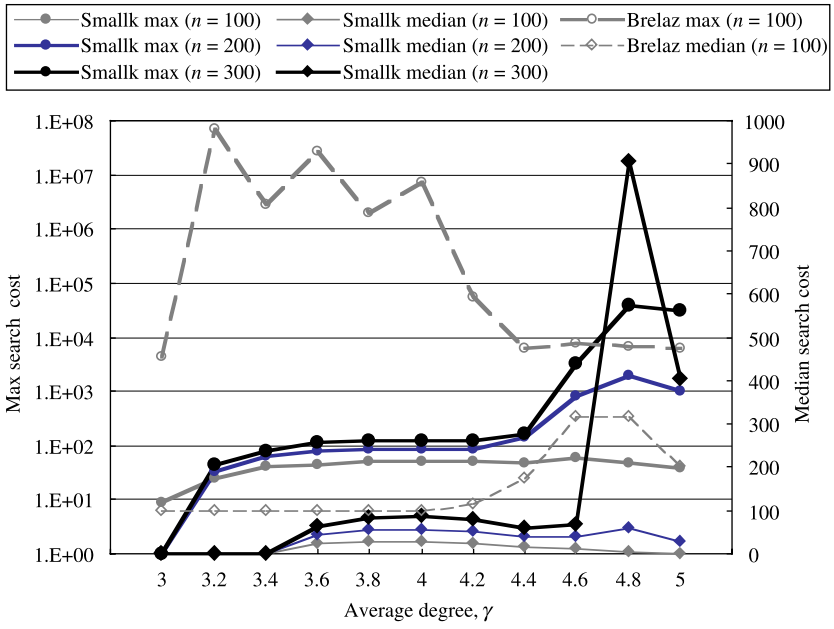

Fig. 8. Experimental results on randomly generated instances.

generated instances, each of which is solved using Smallk. In Brélaz algorithm also, 1.1 million instances with $n = 100$ are used. As shown in Fig. 8, it is obvious that the hardness of our instance set cannot be compared with that of this huge set of randomly generated instances, where, in Smallk, it is only 2.6 and 103 s to determine the colorability of each

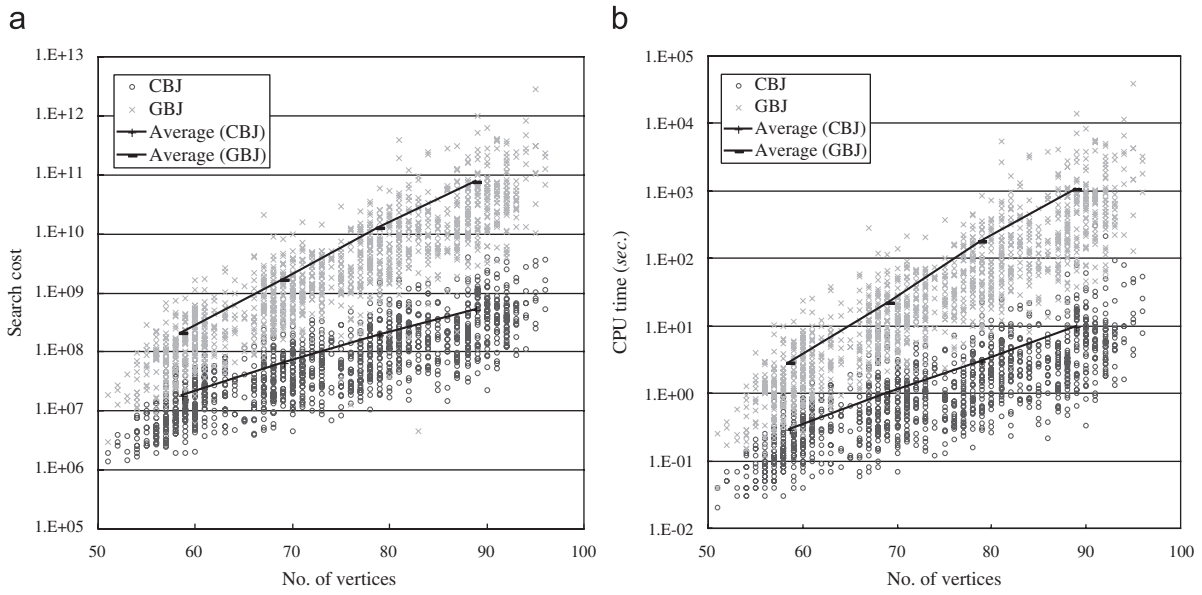

Fig. 9. Experimental results on CBJ and GBJ: (a) search cost; (b) CPU time.

hardest instance with *200* and *300 vertices* at $\gamma = 4.8$, whereas it requires hundreds of seconds on average in solving our instances with even *100 vertices* in Fig. 6.

4.2. Discussion

Experiments confirmed that our method stably and invariably produces very hard instances whose computational cost is of an exponential order of the number of vertices, n. Researchers adopting generate-and-test approaches found that conditions under which hard instances tend to occur are as follows: (1) their constraint density is near the phase transition region, (2) the smallest minimal unsolvable subproblem [14] is very large compared to the instances size, (3) their structure is homogeneous, i.e., quasi-regular. It seems reasonable that instances produced by our method meet all these conditions. Because our instances contain no n4c's, most of the frozen pairs [4] are hidden from the surface, which makes our instances hard to solve even for sophisticated algorithms such as Smallk [19].

We still do not know theoretically why our instances become extraordinarily hard. The ultimate question may be whether our instances are really inherently hard for any other search algorithms. Our instances are 4-critical, meaning deletion of an arbitrary edge changes them to 3-colorable. It also means our instances approach the narrow line between solvable and unsolvable. Experiments affirmed that our method produces solvable hard instances also lying on that line. On the other hand, each of our proposed n4c-free MUGs in Fig. 3 is not triangle free, which seems to be a key issue for 3COL. It may be necessary to find triangle-free MUGs for generating harder solvable/unsolvable instances. We may also need to extend our embedding operation by exploiting known hard structures, e.g., pigeonhole [8], pebbling [1] graphs, etc. Even if we use these structures in our constructive generation algorithm, however, we must take care so that we can generate as 4-critical instances as possible because the lack of 4-criticality can make 3COL instances easier to solve.

Let us move on to a significant issue probably associated with heuristics. Fig. 3 introduces only seven n4c-free 4-critical graphs independent of each other. To produce harder instances, it is indispensable to increase such graphs which are raw materials for our constructive generation method. It is also necessary to ensure that independent n4c-free MUGs exist infinitely. Although we surmise that there are arbitrarily large ones, we still do not know how to generate them systematically. If we find such a method, however, the fact itself may affirm the existence of heuristics. It may hypothetically be the important problem led to the ultimate conclusion to prove whether there are other n4c-free MUGs which cannot be generated even by using any method to systematically generate independent n4c-free MUGs.

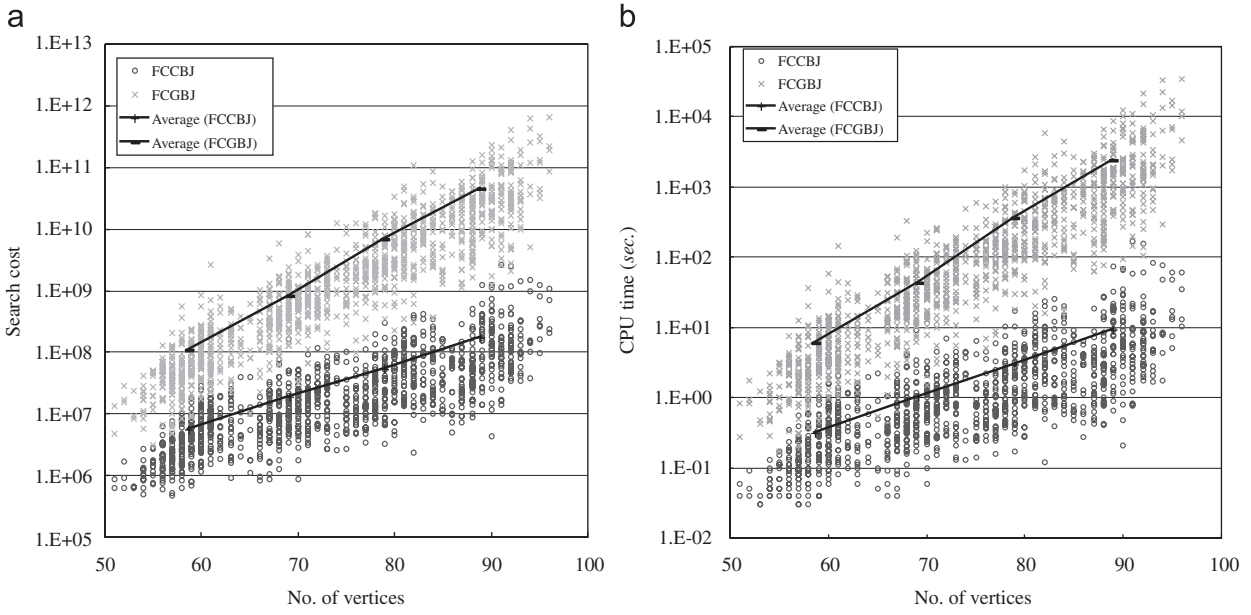

Fig. 10. Experimental results on FCCBJ and FCGBJ: (a) search cost; (b) CPU time.

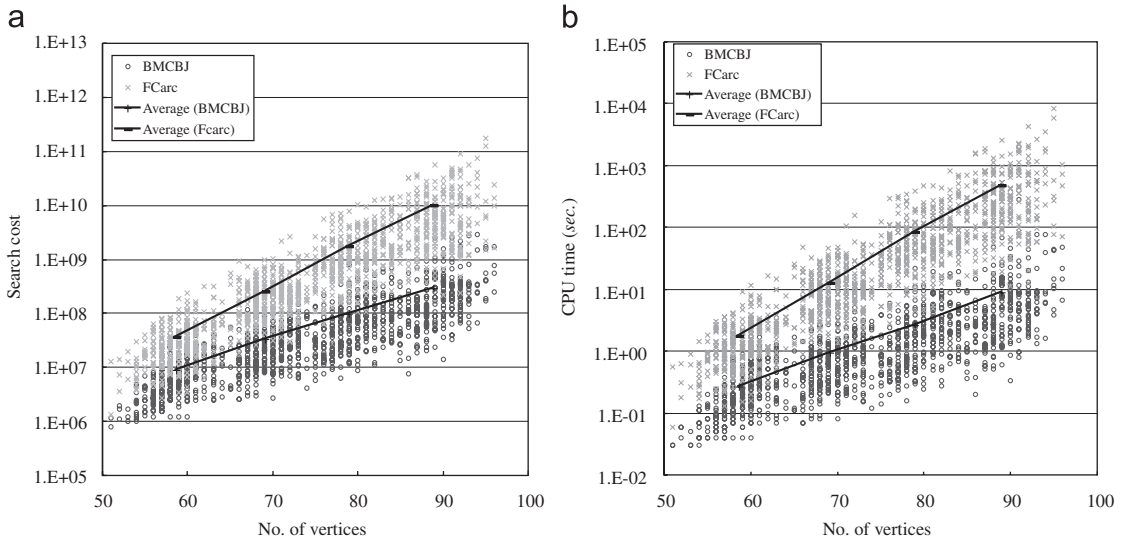

Fig. 11. Experimental results on BMCBJ and FCarc: (a) search cost; (b) CPU time.

4.3. Difficulty for CSP algorithms

To confirm whether our instances are inherently hard, we conduct other experiments, using some well-known CSP algorithms. We apply six major CSP solvers, that is, conflict-directed backjumping (CBJ) [20], graph-based backjumping (GBJ) [5], forward checking with CBJ (FCCBJ) [20], forward checking with GBJ (FCGBJ), backmarking with CBJ (BMCBJ) [20], and arc consistency forward checking with simple backtracking (FCarc) [13], to the same instance set as experiments on Brélaz algorithm in Section 4.1, i.e., 1200 instances from $k = 5$ to $k = 8$.

Figs. 9–11 give experimental results. As well as results of Brélaz algorithm and Smallk in Fig. 6, the computational cost for solving our instances also appears to be quite nearly exponential in the number of vertices even for these CSP solvers, showing that our instances may be intrinsically hard for several search algorithms.

5. Conclusion

We have proposed a constructive algorithm to generate extraordinarily hard instances for graph 3-colorability, which recursively repeat self-embedding operations of our proposed n4c-free MUGs. Instances generated by our method are 4-critical and contain no n4c's, to hide a structural weakness that heuristics would be able to exploit. Using Brélaz algorithm, Smallk, and some well-known CSP solvers, we experimentally showed that the complexity of our 3COL instances is of an exponential order of the number of vertices. Our future works should consist of developing a systematic method to arbitrarily produce many MUGs independent of each other to construct hard instances, clarifying whether heuristics exist that cope with these instances.

Acknowledgments

This research was supported in part by the Ministry of Education, Culture, Sports, Science and Technology of Japan, Grant-in-Aid for Scientific Research (B)(2), No. 14380134, 2002–2005.

References

[1] P. Beame, H. Kautz, A. Sabharwal, Towards understanding and harnessing the potential of clause learning, J. Artif. Intell. Res. 22 (2004) 319–351.
[2] D. Brélaz, New methods to color the vertices of a graph, Comm. ACM 22 (4) (1979) 251–256.
[3] P. Cheeseman, B. Kenefsky, W.M. Taylor, Where really hard problems are, Proceedings of the 12th IJCAI, 1991, pp. 331–337.
[4] J. Culberson, I. Gent, Frozen development in graph coloring, Theoret. Comput. Sci. 265 (2001) 227–264.
[5] R. Dechter, Enhancement schemes for constraint processing: backjumping, learning, cutset decomposition, Artif. Intell. 41 (1990) 273–312.
[6] D. Eppstein, Improved algorithms for 3-coloring 3-edge-coloring, constraint satisfaction, ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 329–337.
[7] S.A. Grant, B.M. Smith, Modelling exceptionally hard constraint satisfaction problems, Proceedings of the CP'97, 1997, pp. 182–195.
[8] V. Guruswami, S. Khanna, On the hardness of 4-coloring a 3-colorable Graph, Electronic Colloquium on Computational Complexity, Report No. 73, 2000.
[9] D. Hanson, G.C. Robinson, B. Toft, Remarks on the graph colour theorem of Hajós, Congr. Numer. 55 (1986) 69–76.
[10] A. Hertz, B. Jaumard, M.P. de Aragão, Local optima topology for the k-coloring problem, Discrete Appl. Math. 49 (1994) 257–280.
[11] T. Hogg, C.P. Williams, The hardest constraint problems: a double phase transition, Artif. Intell. 69 (1994) 359–377.
[12] L. Lovász, Combinatorial Problems and Exercises, North-Holland, Amsterdam, 1979.
[13] A.K. Mackworth, Consistency in networks of relations, Artif. Intell. 8 (1977) 99–118.
[14] D.L. Mammen, T. Hogg, A new look at easy–hard–easy pattern of combinatorial search difficulty, J. Artif. Intell. Res. 7 (1997) 47–66.
[15] K. Mizuno, A. Hayashimoto, S. Nishihara, Analysis of phase transitions in graph-coloring problems based on constraint structures, Proceedings of the PRCAI'2000, 2000, pp. 792.
[16] K. Mizuno, S. Nishihara, Toward ordered generation of exceptionally hard instances for graph 3-colorability, Computational Symposium on Graph Coloring and Its Generalizations (COLOR'02), 2002, pp. 1–8.
[17] K. Mizuno, S. Nishihara, A systematic approach to generate very hard 3COL instances, ISMP'2003, 2003.
[18] S. Nishihara, K. Mizuno, K. Nishihara, A composition algorithm for very hard graph 3-colorability instances, Proceedings of the CP'2003, 2003, pp. 914–919.
[19] Overview of the Smallk Graph Coloring Program, 2000, ⟨http://www.cs.ualberta.ca/~joe/Coloring/smallk.html⟩.
[20] P. Prosser, Hybrid algorithms for the constraint satisfaction problem, Comput. Intell. 9 (1993) 268–299.
[21] B.M. Smith, S.A. Grant, Sparse constraint graphs and exceptionally hard problems, Proceedings of the 14th IJCAI, 1995, pp. 646–654.
[22] D.R. Vlasie, Systematic generation of very hard cases for graph 3-colorability, Proceedings of the 7th IEEE ICTAI, 1995, pp. 114–119.
[23] D.R. Vlasie, The very particular structure of the very hard instances, Proceedings of the 13th AAAI, 1996, pp. 266–270.
[24] T. Walsh, $2 + p$-COL, Computational Symposium on Graph Coloring and Its Generalizations (COLOR'02), 2002, pp. 17–21.
[25] C.P. Williams, T. Hogg, Exploiting the deep structure of constraint problems, Artif. Intell. 70 (1994) 73–117.