Using Hajós' Construction to Generate Hard Graph 3-Colorability Instances*

Sheng Liu[1,2] and Jian Zhang[1]

[1] Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing 100080, China
{lius, zj}@ios.ac.cn
[2] Graduate University, Chinese Academy of Sciences
Beijing 100049, China

Abstract. In this paper we propose a constructive algorithm using constraint propagation to generate 4-critical graph units (4-CGUs) which have only one triangle as subgraph. Based on these units we construct 4-critical graphs using Hajós' join construction. By choosing Grotztsch graph as the initial graph and carefully selecting the edge to be joined, we make sure that the generated graphs are 4-critical and triangle-free. Experiments show that these graphs are exceptionally hard for backtracking algorithms adopting Brélaz's heuristics. We also give some preliminary analysis on the source of hardness.

1 Introduction

Given an undirected graph $G = (V, E)$ with V the set of vertices and E the set of edges, let $|V| = m$ and $|E| = n$. A proper coloring of G is an assignment of colors to vertices such that each vertex receives one color and no two vertices connected by an edge receive the same color. A k-coloring of G is a proper coloring that uses k colors. The smallest number of colors needed to color a graph G is its chromatic number, which is denoted by $\chi(G)$.

Graph coloring problem (GCP) is of great importance in both theory and applications and has been studied intensively in computer science and artificial intelligence (AI). It arises in many applications such as scheduling, timetabling, computer register allocation, electronic bandwidth allocation. However, finding the chromatic number of a given graph is very hard, and even determining whether a given graph can be colored with 3 colors (*3-colorability*) is a standard NP-Complete problem [1], thus being not efficiently solvable by current methods. Despite this, the practical importance of the problem makes it necessary to design algorithms with reasonable computational time to solve instances arising in real-world applications. A lot of work has been done [2, 3, 4] and many powerful techniques have been proposed [5, 6, 7, 8]. On the other hand, in order to

* Supported in part by the National Science Foundation of China (grant No. 60125207).

J. Calmet, T. Ida, and D. Wang (Eds.): AISC 2006, LNAI 4120, pp. 211–225, 2006.

compare and evaluate the performance of different graph coloring algorithms, good benchmarks are needed and thus studied by many researchers.

Providing test instances as benchmarks for AI programs never lacks of interest [9]. Good test instances provide a common reference for people involved in developing and testing new search algorithms. What's more, good benchmarks especially those generated systematically make it possible to take a closer look at the instances and scrutinize their structures, and may give hints for the design of more appropriate algorithms.

Of course, for the sake of better discrimination of different algorithms, among all the test instances, hard ones are preferred. Many researchers discuss the source of hardness of some NP-Complete problems [10, 11]. The mechanism that makes *colorability* very hard is also studied. Possible candidates of order parameters proposed include the 3-path [9], the minimum unsolvable subproblems [12], frozen development [13], etc. Some methods that generate hard real instances are also presented. However, some of them are based on generate-and-test approaches [9], while others use handmade graph units [14], so most of them are either non-deterministic or not repeatable. In this paper we first propose a constructive algorithm that generates small 4-critical graph units (4-*CGU*s), then, similar to [14] we use a recursive self-embedding operation on these 4-*CGU*s to generate big instances. However, our small 4-*CGU*s are generated systematically with the guidance of constraint propagation, not by trial-and-error, so it gives chances to investigate the inner structures of the units and have a good understanding of the source of hardness of the big graphs. And absolutely contrary to [9] which favors graphs having as many triangles as possible, the resulting graphs generated by us are completely triangle-free. Experiments show that our generated graphs are very hard to solve for backtracking algorithms adopting Brélaz's heuristics [5] such as Trick[1].

The outline of the paper is as follows. In the next section we give some notations on constraint propagation and propose an algorithm that creates 4-*CGU*s. In section 3, we introduce the Hajós' join construction and generate big 4-critical graphs with it. Experimental details and analysis are listed in section 4. Comparisons with related work are given in section 5 and in section 6 we conclude the paper.

2 Constraint Propagation and 4-*CGU*s Generation

As mentioned above, what we generated are *3-colorability* instances. However, 4-critical graphs have the property that they are 3-colorable if any vertex/edge is removed (we denote them as vertex-critical graphs and edge-critical graphs respectively), so we first generate a 4-critical graph and when a 3-colorable graph is needed we simply remove some vertex/edge. The graphs that we consider and generate in this article are edge-critical.

For the sake of completeness, we recall some basic notations and definitions of the constraint satisfaction problems (CSPs) [15].

[1] http://mat.gsia.cmu.edu/COLOR/color.html

A CSP consists of a set of n variables $X_1, X_2, \ldots, X_n$ and a set of n domains $D_1, D_2, \ldots, D_n$ where each D_i defines the set of values that the variable X_i may assume. A solution of a CSP is an assignment of a value to each variable which satisfies a given set of constraints. A binary CSP is one in which all constraints involve only pairs of variables. A binary constraint between X_i, X_j is a subset of the Cartesian product $D_i \times D_j$. A binary CSP can be associated with a constraint graph in which vertices represent variables and edges connect pairs of constrained variables.

Definition 1. *A constraint graph is* ***Arc Consistent*** *iff for each of its arcs* $< i, j >$ *and for any value* $a_i \in D_i$*, there is a value* $a_j \in D_j$ *that satisfies the binary constraint between* i *and* j*.*

GCP is to assign colors to vertices of a graph with constraints over its edges, so the constraint graph can be obtained directly and easily. In fact, there is a one-to-one mapping between a graph to be colored and its constraint graph, in which vertices correspond to variables and colors to be assigned to a vertex correspond to the domain of the corresponding variable. So many papers on constraint processing take graph coloring problems as examples, and constraint satisfaction thus promotes the research on graph coloring. Our work is carried out with the guidance of constraint satisfaction. Now we propose our algorithm **CGU(4,n)** which constructs 4-CGU with n vertices ($n \geq 9$).

Algorithm 1. CGU(4,n)

Step 1 Let $n = 3 * m + r$ where both m and r are non-negative integers, and $r < 3$.

Step 2 Construct a triangle $\triangle ABC$ and a circle with $3 * (m-1)$ vertices denoted as $a_1, b_1, c_1, a_2, b_2, c_2, \ldots, a_{m-1}, b_{m-1}, c_{m-1}$ successively.

Step 3 Connect A with all a_i (i=1, 2, ..., m-1);
connect B with all b_i (i=1, 2, ..., m-1);
connect C with all c_i (i=1, 2, ..., m-1).

Step 4 (a) If r=0 then choose two vertices a_k, a_l from a_i (i=1, 2, ..., m-1), connect a_k and a_l;
(b) if r=1 then choose a vertex a_k from a_i (i=1, 2, ..., m-1), a vertex b_l from b_i (i=1, 2, ..., m-1) and a vertex c_m from c_i (i=1, 2, ..., m-1), introduce a new vertex O, connect O with a_k, O with b_l, O with c_m;
(c) if r=2 then choose two vertices a_{k_1}, a_{k_2} from a_i (i=1, 2, ..., m-1), choose two vertices b_{l_1}, b_{l_2} from b_i (i=1, 2, ..., m-1), introduce two new vertices O_1, O_2, connect O_1 with a_{k_1}, O_1 with b_{l_1}, O_2 with a_{k_2}, O_2 with b_{l_2}, O_1 with O_2;

Step 5 Stop.

Note that each graph generated by the algorithm **CGU(4,n)** has a triangle in it. So the chromatic number is at least 3. In fact we have the following theorems:

Theorem 1. *The graphs constructed by* ***CGU(4,n)*** *are 4-chromatic.*

Proof. For the sake of convenience, without loss of generality we generate a small graph and take it as an example (Fig. 1).

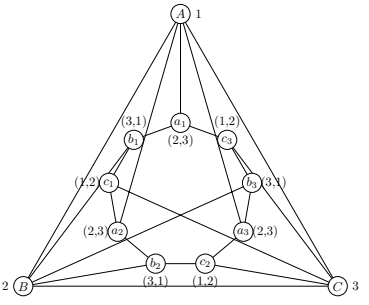

Fig. 1. Part of the graph after the triangle is colored

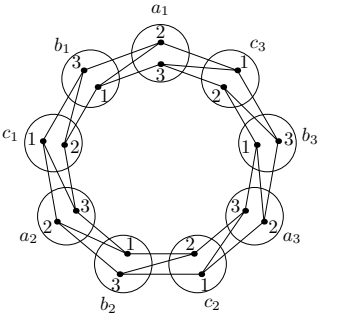

Fig. 2. Part of the constraint graph on the circle

We prove it by contradiction. Assume that it is 3-colorable. As depicted in Fig. 1, because there is a triangle $\triangle ABC$ in the graph, the three vertices A, B, C of the triangle can be colored by 3 colors denoted as color $1, 2, 3$. Then by constraint propagation we get that the vertices on the circle that are connected with vertex A can only be colored by color 2 or color 3. Similarly, the vertices on the circle that are connected with vertex B (C) can only be colored by color 3 or color 1 (color 1 or color 2). Thus we get a circle with candidate colors in every vertex (Fig. 1). Because the edges on the circle can also denote constraints, we apply arc consistency test on each edge and get part of the constraint graph shown in Fig. 2[2]. Furthermore,we notice that each of the candidate colors of the vertices in Fig. 2 has one edge pointing to one of its two neighbors and two edges pointing to the other neighbor, so once the color of one vertex is fixed, one of its neighbors' color is fixed at the same time. For instance, if we set color 3 to vertex a_1, then vertex b_1 can only be colored with color 1. In the same way vertex c_1 can only be colored with color 2, and the rest may be deduced by analogy. It is the same when vertex a_1 chooses color 2. Thus no matter what color a_1 chooses, once its color is fixed, the colors of all the other vertices are also fixed. Meanwhile it is interesting that all the vertices connected with the same vertex of the triangle have the the same color but vertices connected with different vertices of the triangle have different colors. For instance, all a_i (i=1, 2, ..., m-1) have the same color but b_i (i=1,2, ...,m-1) and c_i (i=1,2, ...,m-1) enjoy different colors.

Next we discuss the construction in **Step 4** (Fig. 3, Fig. 4, Fig. 5). **(*a*)** If r=0 then one edge between a_k and a_l is constructed. But from the discussion above we know that a_k and a_l should have the same color, so it makes a contradiction. However, one more color assigned to a_k or a_l (but not both) is enough to solve the contradiction. **(*b*)** If r=1 then because O is connected with a_k, b_l and c_m

[2] For the sake of readability, we do not depict the complete constraint network of the vertices on the circle. In fact there are many edges between the vertex a_1 and each of its non-neighbor vertices, but we neglect such edges for simplicity.

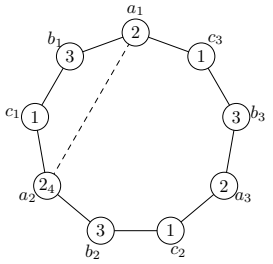

Fig. 3. r=0

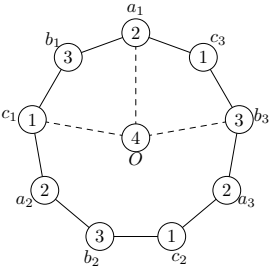

Fig. 4. r=1

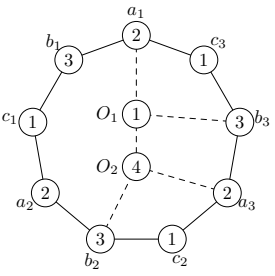

Fig. 5. r=2

which have mutually different colors, O can't be colored by any of the three colors. So a fourth color is needed to color O. **(*c*)** If r=2 then because O_1 is connected with a_{k_1} and b_{l_1}, it can't have the same color as a_{k_1} or b_{l_1}, thus it has to choose the color assigned to c_i (i=1, 2, ..., m-1). It is the same for O_2. But O_1 and O_2 are also connected, so they can't get the same color at the same time. It contradicts again and one more color is needed.

To sum up, the graphs constructed by **CGU(4,n)** are not 3-colorable but 4 colors are adequate to color them, so they are 4-chromatic. □

Theorem 2. *The graphs constructed by **CGU(4,n)** are 4-critical.*

Proof. Now that it has been proved in Theorem 1 that graphs constructed by **CGU(4,n)** are 4-chromatic, we only need to prove that the graphs are 3-colorable if an arbitrary edge is removed, according to the definition of critical graph.

From the analysis above we know that there indeed exists a coloring scheme using 4 colors in which only one vertex (i.e., O in Fig. 4) is colored with the fourth color. Our proof begins with such a scheme. Once an edge is removed from the graph, we prove that by changing the original 4 coloring scheme step by step we can get a new coloring scheme which uses only 3 colors, that is to say, the newly introduced color (the fourth color) can be replaced by one of the original 3 colors because of the removal of one edge.

First we study the edges generated in **Step 4** of the algorithm **CGU(4,n)** (the dashed edges in Fig. 3, Fig. 4 and Fig. 5). Because all these edges are adjacent to the vertex colored by the fourth color[3], once such an edge is removed, the end vertices of that edge can share the same color. So the fourth color is not indispensable any more.

Next we consider the edges forming the circle and the edges between the triangle and the circle. From Fig. 3, Fig. 4 and Fig. 5 we find that the dashed edges divide the region inside the circle into two or more subregions, each of

[3] In the case of r=2, only one of O_1 and O_2 has to be colored by the fourth color, so if O_1 is colored by the fourth color it seems that the judgment does not hold for the edges that are adjacent to O_2 but not adjacent to O_1. But since the color of O_1 and the color of O_2 can be exchanged, it does not affect the following discussion and conclusions.

which is circumscribed by a closed subcircle (e.g., (a_1, b_1, c_1, O, a_1) of Fig. 4). Each edge of the original circle is on one of these subcircles and each subcircle contains the vertex that is colored by the fourth color[4]. If one of the edges on the circle is removed (e.g., edge (b_2, c_2) of Fig. 4), the constraint over (b_2, c_2) does not exist any more, so vertex b_2 and c_2 can have the same color. Right now b_2 has color 3 and c_2 has color 1, but we can't assign b_2's color 3 to c_2 because c_2 is connected with the triangle vertex C whose color is also 3, so we have no other choice but to assign c_2's color 1 to b_2. Then we propagate the color assignment along the subcircle successively, starting from b_2's color and ending up when we get to O. In this sample, we assign b_2's former color 3 to a_2, assign a_2's former color 2 to c_1, assign c_1's former color 1 to O and at last O's former color 4 is discarded. So far the graph has been colored by 3 colors without any color collisions. In case that the removed edge is one of the edges between the triangle and the circle (i.e., edge (B,b_2)), similarly, we can assign B's color 2 to b_2. But one of b_2's neighbors a_2 on the circle also has color 2 at present, so we first color a_2 with b_2's former color 3 and then propagate the color assignment as we described above, in the direction from b_2 to a_2 along the subcircle. Thus the graph is colored by 3 colors properly.

At last we turn to the edges that form the triangle. If one of the triangle edges (e.g., (A, B)) is removed, the endpoints (A and B) of the edge can share the same color. Since all the a_i-form and b_j-form vertices receive the same color constraint from triangle vertices, it follows that they are equivalent in fact. Thus each a_i-form vertex can change colors with its b_j-form neighbor, vice versa. As for the vertex that is colored by the fourth color (i.e., O'), it has at least one a_i-form or b_j-form neighbor. Next we first discard color 4 and color O' with the color of its a_i-form or b_j-form neighbor (i.e., b_k). Then a conflict arises because the two vertices O' and b_k are connected by an edge but have the same color. In order to overcome the conflict, we first exchange colors between b_k and its a_i-form neighbor on the circle (i.e., a_k). If this leads to a new conflict between b_k (a_k) and B (A), we need only let B (A) have the same color as A (B). Take Fig. 5 for example, first we change the color of O_2 from 4 to 3, after that we exchange colors between a_2 and b_2. This leads to a new conflict between b_2 and B, so we assign A's color 1 to B, then we get a proper 3-coloring of the graph.

After checking all the cases, we reach the conclusion that the generated graph is 3-colorable no matter which edge is removed. Thus Theorem 2 is proved. □

3 Hajós' Join Construction and Hard Triangle-Free Graph Generation

As mentioned in the introduction, starting with small 4-CGUs, we use Hajós' construction to build big critical graphs. So we introduce Hajós' join construction first [16].

Definition 2 (The Hajós' construction). *Let G and H be two graphs. Let uu' be an edge in G and vv' be an edge in H. The resulting graph $G \triangle H$ is*

[4] The case of r=2 has been discussed above.

obtained by identifying u and v, deleting the edges uu' and vv', and adding an edge $u'v'$.

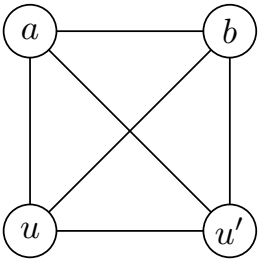

Fig. 6. Graph G

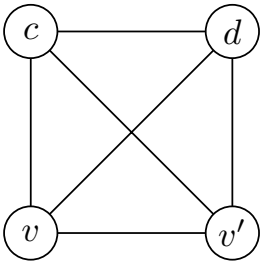

Fig. 7. Graph H

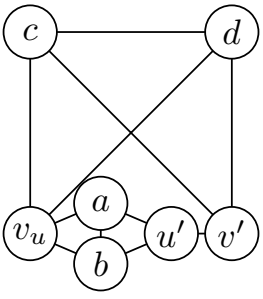

Fig. 8. Graph $G \triangle H$

Theorem 3 (Hajós). *If both G and H are k-critical graphs ($k > 3$), then $G \triangle H$ is a k-critical graph.*

Now we can use Hajós' construction iteratively to construct big critical graphs without altering their chromatic number. Details are listed below:

procedure $HardGraph(k)$
begin
  $G := G_{init}$;
  for $i := 1$ **to** k **do**
    choose a random number l ($l \geq 9$);
    $H :=$**CGU(4,l)**;
    $G := Hajos(G, H)$;
  od;
end
procedure $Hajos(G, H)$
begin
  choose an edge uu' from G and remove it;
  choose an edge vv' from H and remove it;
  add edge $u'v'$;
  merge u with v;
end

According to Hajós' construction, in order to generate 4-critical graphs the graph G_{init} used in $HardGraph(k)$ must also be 4-critical. [14] finds 7 MUGs (minimal unsolvable graphs, which are also small 4-critical graphs) by trial-and-error and chooses one of them as the initial graph G_{init}. Some MUGs contain more than one triangles, so the resulting instances may contain many triangles. But by choosing a_{k_1} not adjacent to b_{l_1} and a_{k_2} not adjacent to b_{l_2} in **CGU(4,n)**, we easily make sure that each of the 4-CGUs generated by our algorithm has only one triangle. In $Hajos(G, H)$ we choose one of the three edges of the triangle in

H as the joining edge vv', then the remaining part of H is triangle free. Since $Hajos(G, H)$ doesn't introduce any new triangles into the generated graph, if we choose a triangle-free graph as the initial graph G_{init} we can make sure that all our generated graphs are triangle-free. So we use the triangle-free Grotztsch graph as the initial graph (Fig. 9).

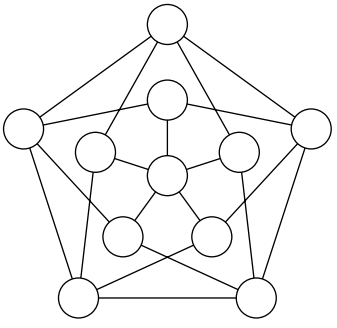

Fig. 9. Grotztsch graph

The size of maximum clique is usually used as a rough lower bound of the chromatic number. Some researchers speculate that the greater the distance between the chromatic number and the lower bound is, the harder the graph is for algorithms that color graphs by detecting lower bound first [17]. Because the maximum clique in triangle-free graphs is of size 2, while the size of that in non-triangle-free graphs is at least 3, we guess that our triangle-free graphs may be harder and our experimental results support our speculation to some extent. We will describe them in the next section in detail.

4 Experiments and Discussion

After the description of the algorithms in section 2 and section 3, we devote this section to some implementation details.

4.1 Generating Better 4-*CGU*s

In **Step 4** of algorithm **CGU(4,n)**, some vertices on the circle such as a_k and b_l have to be chosen. However, there are many choices for these vertices so we have to decide the relative better ones. Of course we prefer choosing vertices that make the generated 4-*CGU*s harder to solve. We compare two versions of implementation. In $\mathbf{CGU}_1$**(4,n)** all the chosen vertices are distributed relatively densely on the circle while in $\mathbf{CGU}_2$**(4,n)** they are distributed as uniformly as possible. We believe that more subproblems lead to more backtracks when the graph is colored by Trick. So we record the number of subproblems when applying Trick to color a graph, which is listed in Table 1. All the experiments were carried out on a P4 2.66GHz Dell computer with 512M memory.

Table 1. Comparison between two kinds of **CGU(4,n)**s on subproblem numbers

n	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$\mathbf{CGU_1(4,n)}$	12	13	16	13	16	21	16	19	27	19	22	33	22	25	39	25	28
$\mathbf{CGU_2(4,n)}$	12	13	16	16	19	22	19	22	28	25	28	34	28	34	39	34	37

From the table, we notice that there is not much difference between $\mathbf{CGU_1(4,n)}$ and $\mathbf{CGU_2(4,n)}$ when r=2 ($n = 11, 14, 17, 20, 23, \ldots$). However, in other cases, no matter what value n is assigned to, $\mathbf{CGU_2(4,n)}$ performs much better than $\mathbf{CGU_1(4,n)}$. We also generated some graphs using $\mathbf{CGU_1(4,n)}$ and $\mathbf{CGU_2(4,n)}$ respectively. Experimental results show that averagely graphs using $\mathbf{CGU_2(4,n)}$ are harder to color and all the hardest ones among all the generated graphs are those using $\mathbf{CGU_2(4,n)}$. The superiority of $\mathbf{CGU_2(4,n)}$ is evident so we adopt the $\mathbf{CGU_2(4,n)}$ version in our experiment.

4.2 Making Structure More Regular

From algorithm **CGU(4,n)** we find that all the vertices except the ones on the triangle have lower variance of degrees. However, the degrees of the triangle vertices increase quickly as n increases. In [9] the author finds that more regular instances, with more uniform structures, tend to be much harder. We also generated two kinds of graphs. One kind uses **CGU(4,n)**s with n ranging from 9 to 15 and the other kind uses **CGU(4,n)**s with n ranging from 16 to 22. Experiments show that the latter ones are not so hard as the former ones. As a matter of fact some of the latter graphs are not hard at all although they have more vertices. It seems that regularity[5] is an important factor in the hardness of graphs.

On the other hand, when we use Hajós' construction, the degree of vertex v also increases because of merging u with v. So in order to prevent the degree of v from increasing too much, when choosing the edge vv' we deliberately choose the vertex with degree 3 as v. We also compare such generated graphs with the ones generated by selecting v randomly, denoted by asterisk $*$ and plus $+$ respectively. Figure 10 depicts the average search cost comparison for each n and Fig. 11 depicts the maximal search cost comparison. From the comparison results we find that restricting vertices' degrees to lower variance indeed makes instances harder. It seems that regular instances induce uniformity in the structure of the search space, making the search algorithm confused by the equally promising choices [9]. So in the following experiments we use **CGU(4,n)**s with n ranging from 9 to 15 and deliberately choose a vertex with degree 3 as v, as we did above.

4.3 Generating Hard Triangle-Free Graphs

With **CGU(4,n)**s (n ranges from 9 to 15) and Grotztsch graph we generate triangle-free graphs using $HardGraph(k)$ where k ranges from 5 to 12. For each

[5] A graph is regular if all its vertices have the same degree.

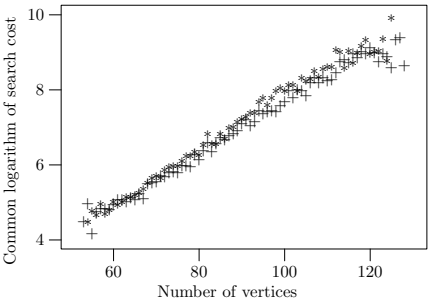

Fig. 10. Comparison on average search cost

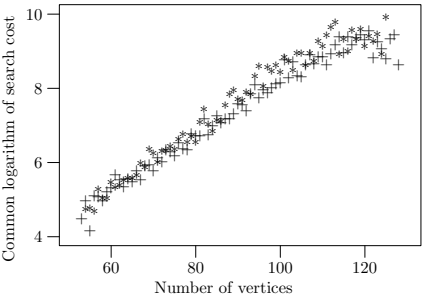

Fig. 11. Comparison on maximal search cost

value of k, 200 instances are generated. These instances are tested by Trick and the results are given in Fig. 12 in which search cost is evaluated by the number of subproblems. Figure 12 reveals a linear relationship between the vertical axis and the horizontal axis. However, note that the vertical axis of Fig. 12 represents common logarithm of the number of subproblems, so it is easy to understand that the search cost exhibits exponential growth.

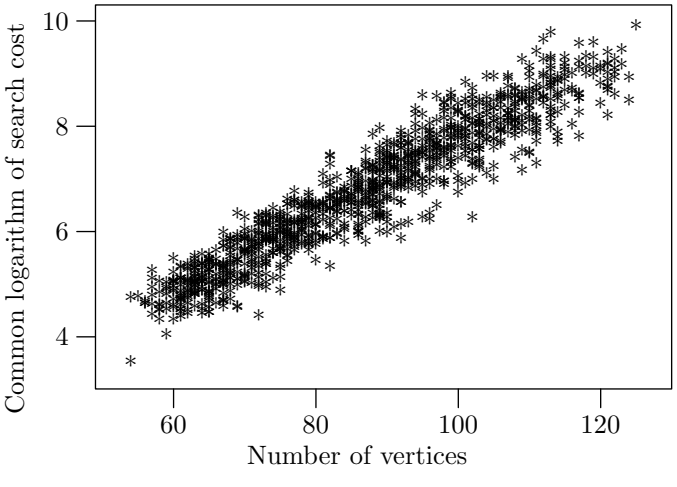

Fig. 12. Experimental results of the triangle-free instances

We also generate instances from [14] and compare them with our triangle-free instances. Figure 13 depicts the average search cost comparison for each n and Fig. 14 depicts the maximal search cost comparison. Here, the asterisk $*$ corresponds to our data while the plus $+$ corresponds to data of [14]. Figure 13 reveals that when n is small, instances of [14] seem superior, but when n grows bigger and bigger our triangle-free instances turn out to be superior. The same

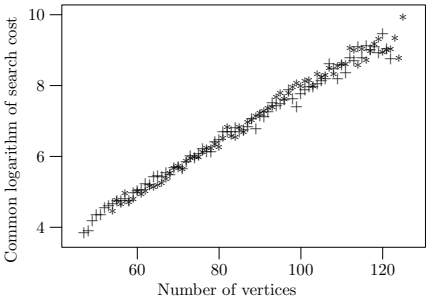

Fig. 13. Comparisons on average search cost

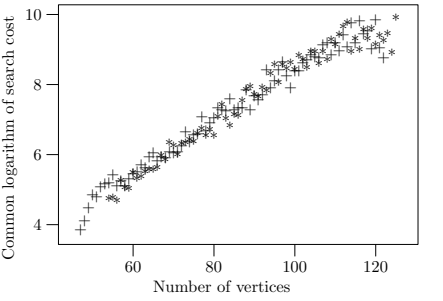

Fig. 14. Comparisons on maximal search cost

result can be obtained from Fig. 14 except that it is not so evident as in Fig. 13. We speculate that the tendency may go on, but it is hard to verify it by further experiment. On one hand, for the sake of statistical accuracy, for each vertex number n we need the sample space as big as possible, on the other hand, when n grows bigger and bigger, each sample needs so much time that the total time cost is too much. However, current results have already shown that our triangle-free instances are at least as hard as and maybe harder than those of [14].

We also compare our instances with those of [14] by running other graph coloring algorithms. Next we give our experimental results with Smallk [13], a sophisticated backtracking coloring program specialized for graphs of small chromatic number.

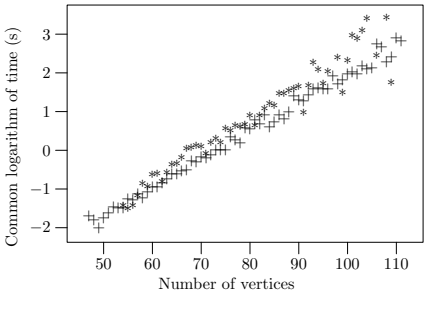

Fig. 15. Comparisons with Smallk on average time

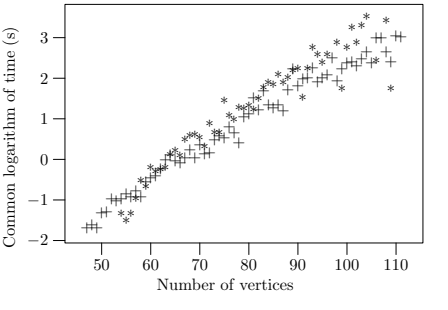

Fig. 16. Comparisons with Smallk on maximal time

From Fig. 15 and Fig. 16 we find that although both kinds of instances exhibit exponential growth, our triangle-free ones seem to be a little harder in general. It is known that Smallk is good at exploiting structural weakness (e.g., frozen pairs in [13]). Maybe our instances just hide such weakness because they are triangle-free.

In summary, our experiments show that our construction is efficient in producing very hard instances whose computational costs seem to be of an exponential order of the vertex number.

4.4 Analysis and Discussion

Decomposition and composition are widely used in graph coloring except that decomposition usually deals with complicated graphs by dividing them into small components and analyzing the small components while composition usually builds complicated graphs from small components. Experiments in [14] and here show that the Hajós' construction is an effective composition method to generate hard *3-colorability* instances. We also notice that, in [14], because the graph components MUGs were found totally by trial-and-error, the authors don't have a good knowledge of the components' structural features, so that they can not explain the reason of the resulting graphs' hardness. However our component 4-*CGU*s are generated systematically with the guidance of constraint propagation, so by scrutinizing their inner structures, we can give some useful information in understanding the reason.

In our construction each 4-*CGU* is a component and it is iteratively embedded into the resulting instance. From the proof of Theorem 1 we know that for the vertices on the circle (Fig. 2) once one of the vertices is colored the other vertices' colors are fixed at the same time. That is to say, only one choice remains for each of the other vertices. This conclusion can be obtained by constraint propagation as described in this paper, but for many algorithms adopting backtracking heuristics they can't foresee the full future, so it is inevitable to make wrong decisions. What's more, wrong decision can occur in every 4-*CGU*. Notice that our instances are generated by iteratively embedding 4-*CGU*s which can also be viewed as the multiplying of 4-*CGU*s. So, even if there is one backtrack in each 4-*CGU*, there will be an exponential number of backtracks altogether in the resulting graphs. For this reason backtracking algorithms will spend exponential time detouring and backtracking before they find the right coloring. So it seems exceptionally hard for them to handle our instances.

5 Related Work

A lot of work has already been done on providing benchmarks for general GCP, but our work focuses on graph *3-colorability*.

Related work on generating hard graph *3-colorability* instances includes [14] and [9]. [14] also uses Hajós' construction. But, in order to generate very hard instances, one must have lots of small 4-*CGU*s at hand first[6]. [14] finds 7 such units by trial-and-error, which shows to some extent the difficulty of finding 4-*CGU*s by hand and the necessity of generating them systematically. So, in

[6] What's more, among all 4-*CGU*s the ones including no *near*-4-*clique* (4-*clique* with an edge removed) as subgraphs are preferred, because such graphs hide a structural weakness that heuristics would be able to exploit (e.g., frozen pairs in [13]).

this paper, we present a constructive algorithm to find 4-*CGU*s systematically. What's more, because our 4-*CGU*s are generated automatically with the guidance of constraint propagation, it provides a possibility to investigate the inner structures of the graphs and find some useful rules why they can produce hard graphs. Based on constraint propagation we give some explanations in Section 4, which may help recognize the reason of the hardness and give some hints for researchers working on new coloring heuristics. While in [14] the authors don't have a good knowledge of the reason of the hardness since even their MUGs are found by trial-and-error.

[9] uses a generate-and-test method to produce *3-colorability* instances. The author takes a random graph with fixed vertex number and edge number as an input, then removes an old edge and adds a new edge to the graph iteratively, hoping to minimize the number of 3-paths (denoted by an alternate succession of vertices and edges $x_1e_1x_2e_2x_3e_3x_4, x_1 \neq x_4$). It is easy to find that [9] favors graphs that have as many triangles as possible. Although it avoids 4-clique during construction, it does not avoid *near-4-clique* which appears to be a structural weakness. However, our generated instances do not have such weakness because we make sure that they are all triangle-free by using Grotztsch graph as initial graph and selecting special edges to join. Experiments show that our instances seem to be even harder when tested with sophisticated algorithms such as Smallk.

Although our instances are of small chromatic numbers, they can also be used as general GCP benchmarks. As far as we know, there are already such benchmarks in the DIMACS 2002 Challenge[7] (i.e., mug88_1, mug88_25, mug100_1, mug100_25 are such ones provided by the second author of [14]). We also notice that among the benchmarks in the DIMACS 2002 Challenge, there is a special kind of graphs named Myciel graphs which are based on the Mycielski transformation. Although the chromatic numbers of these graphs range from 4 to 5, 6 and even more, their maximum clique numbers remain 2. Because they are triangle-free, these graphs are difficult to solve. As for chromatic number 3, however, as far as we know, it seems that there are few benchmarks with the same property. But our generated instances (with an edge removed) just have the property. What's more, there are only a fixed number of Myciel graphs for each chromatic number, but many instances can be generated using our method.

6 Conclusions and Future Work

In this paper, a constructive algorithm that generates 4-*CGU*s systematically is presented. With these 4-*CGU*s we generate 4-critical and triangle-free graphs using Hajós' construction. Experiments show that our instances are exceptionally hard for backtracking algorithms adopting Brélaz's heuristics. Because our instances are triangle-free which hides some structural weakness, compared with similarly generated instances, they seem to be harder when experimented with sophisticated backtracking algorithms.

[7] http://mat.gsia.cmu.edu/COLOR02/

As benchmarks, hard instances are good but we believe that hard instances with known inner structures are better, because they can give some hints for researchers working on new coloring heuristics. Our triangle-free instances just have the property. Since our 4-CGUs are generated systematically with the guidance of constraint propagation, we have a good knowledge of their inner structures, which makes it possible for us to give some explanations on the hardness of the resulting 4-critical graphs. We think that one of our contributions is that we present such a constructive algorithm to produce 4-CGUs systematically. We plan to find more methods to produce 4-CGUs with more sophisticated structures, so as to get more knowledge of the inner structures of the generated hard graphs and develop heuristics to solve them.

Acknowledgments

Thanks to Jin-Kao Hao for offering us the executable that helps to verify our proof. Thanks to the anonymous referees for the valuable and detailed comments.

References

[1] Garey, M.R., Johnson, D.S.: Computers and Intractability - A Guide to the Theory of NP-Completeness. W. H. Freeman, San Francisco (1979)
[2] Kubale, M., Jackowski, B.: A generalized implicit enumeration algorithm for graph coloring. Commun. ACM **28**(4) (1985) 412–418
[3] Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. INFORMS Journal on Computing **8** (1996) 344–354
[4] Johnson, D.S., Trick, M.A., eds.: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993. American Mathematical Society, Boston, MA, USA (1996)
[5] Brélaz, D.: New methods to color the vertices of a graph. Commun. ACM **22**(4) (1979) 251–256
[6] Peemöller, J.: A correction to Brélaz's modification of Brown's coloring algorithm. Commun. ACM **26**(8) (1983) 595–597
[7] Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing **39**(4) (1987) 345–351
[8] Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. **3**(4) (1999) 379–397
[9] Vlasie, R.D.: Systematic generation of very hard cases for graph 3-colorability. In: Proceedings of 7th IEEE ICTAI. (1995)
[10] Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: Proceedings of the 12th IJCAI. (1991) 331–337
[11] Hogg, T., Williams, C.P.: The hardest constraint problems: A double phase transition. Artificial Intelligence **69** (1994) 359–377
[12] Mammen, D.L., Hogg, T.: A new look at the easy-hard-easy pattern of combinatorial search difficulty. Journal of Artificial Intelligence Research **7** (1997) 47–66
[13] Culberson, J., Gent, I.: Frozen development in graph coloring. Theoretical Computer Science **265**(1–2) (2001) 227–264

[14] Nishihara, S., Mizuno, K., Nishihara, K.: A composition algorithm for very hard graph 3-colorability instances. In: Proceedings of the 9th CP. Volume 2833 of Lecture Notes in Computer Science. (2003) 914–919
[15] Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
[16] Jensen, T.R., Toft, B.: Graph Coloring Problems. Wiley, New York (1995)
[17] Caramia, M., Dell'Olmo, P.: Constraint propagation in graph coloring. Journal of Heuristics **8**(1) (2002) 83–107