May 19, 2018 at 02:30

1. Intro. This little program outputs clauses that are satisfiable if and only if the graph g can be c-colored, given g and c. It differs from SAT-COLOR because it uses binary labels instead of unary labels to specify the colors.

(It generalizes SAT-PIGEONS-LOG, which is the case where $g = K_m$ and $c = n$.)

Suppose the graph has m edges and n vertices, and let $t = \lceil \lg c \rceil$. Then there are nt variables $v.k$, meaning that vertex v gets color $(v.1\ v.2\ \ldots\ v.k)_2$. There also are mt auxiliary variables $u\hat{}vk$, meaning $u.k \oplus v.k$ when $u \mathrel{—} v$.

There are m clauses of size t to ensure that adjacent vertices don't share a color, plus $4mt$ clauses to define the auxiliary variables. And finally, there are $O(nt)$ additional clauses of size $\le t$, to rule out cases where a vertex is assigned to colors s in the range $n \le s < 2^t$.

```
#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"
  int c;
  main(int argc, char *argv[])
  {
    register int i, k, t;
    register Arc *a;
    register Graph *g;
    register Vertex *u, *v;
    ⟨ Process the command line 2 ⟩;
    for (t = 0; c > (1 ≪ t); t++) ;
    ⟨ Generate negative clauses to rule out bad colors 3 ⟩;
    for (v = g→vertices; v < g→vertices + g→n; v++)
      for (a = v→arcs; a; a = a→next) {
        u = a→tip;
        if (u < v) ⟨ Generate clauses to keep u and v from having the same color 4 ⟩;
      }
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
  if (argc ≠ 3 ∨ sscanf(argv[2], "%d", &c) ≠ 1) {
    fprintf(stderr, "Usage:␣%s␣foo.gb␣c\n", argv[0]);
    exit(−1);
  }
  g = restore_graph(argv[1]);
  if (¬g) {
    fprintf(stderr, "I␣couldn't␣reconstruct␣graph␣%s!\n", argv[1]);
    exit(−2);
  }
  if (c ≤ 0) {
    fprintf(stderr, "c␣must␣be␣positive!\n");
    exit(−3);
  }
  printf("~␣sat-color-log␣%s␣%d\n", argv[1], c);
```

This code is used in section 1.

3. ⟨ Generate negative clauses to rule out bad colors 3 ⟩ ≡

```
for (i = 0; i < t; i++)
  if (((c − 1) & (1 ≪ i)) ≡ 0) {
    for (v = g→vertices; v < g→vertices + g→n; v++) {
      printf("~%s.%d", v→name, t − i);
      for (k = i + 1; k < t; k++)
        if ((c − 1) & (1 ≪ k)) printf("␣~%s.%d", v→name, t − k);
      printf("\n");
    }
  }
```

This code is used in section 1.

4. ⟨ Generate clauses to keep u and v from having the same color 4 ⟩ ≡

```
{
  for (k = 1; k ≤ t; k++) {
    printf("~%s^%s%d␣%s.%d␣%s.%d\n", u→name, v→name, k, u→name, k, v→name, k);
      /∗ printf("%s^%s%d␣%s.%d␣~%s.%d\n", u→name, v→name, k, u→name, k, v→name, k); ∗/
      /∗ printf("%s^%s%d␣~%s.%d␣%s.%d\n", u→name, v→name, k, u→name, k, v→name, k); ∗/
    printf("~%s^%s%d␣~%s.%d␣~%s.%d\n", u→name, v→name, k, u→name, k, v→name, k);
  }
  for (k = 1; k ≤ t; k++) printf("␣%s^%s%d", u→name, v→name, k);
  printf("\n");
}
```

This code is used in section 1.

5. Index.

a: 1.
Arc: 1.
arcs: 1.
argc: 1, 2.
argv: 1, 2.
c: 1.
exit: 2.
fprintf: 2.
g: 1.
Graph: 1.
i: 1.
k: 1.
main: 1.
name: 3, 4.
next: 1.
printf: 2, 3, 4.
restore_graph: 2.
sscanf: 2.
stderr: 2.
t: 1.
tip: 1.
u: 1.
v: 1.
Vertex: 1.
vertices: 1, 3.

⟨ Generate clauses to keep u and v from having the same color 4 ⟩ Used in section 1.
⟨ Generate negative clauses to rule out bad colors 3 ⟩ Used in section 1.
⟨ Process the command line 2 ⟩ Used in section 1.

SAT-COLOR-LOG

	Section	Page
Intro	1	1
Index	5	3