May 19, 2018 at 02:31

1*. Intro. This program generates clauses that enforce the constraint $x_1 + \cdots + x_n \le r$, using a method due to Olivier Bailleux and Yacine Boufkhad [*Lecture Notes in Computer Science* **2833** (2003), 108–122]. It introduces at most $(n-2)r$ new variables Bi.j for $2 \le i < n$ and $1 \le j \le r$, and a number of clauses that I haven't yet tried to count carefully, but it is at most $O(nr)$. All clauses have length 3 or less.

This version inputs a graph (specified as a third parameter) and and color number (specified fourth). The output clauses will limit the number of vertices of that color.

```
#define nmax 10000
#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"
  int n, r, kk;    /* the given parameters */
  int count[nmax + nmax];    /* the number of leaves below each node */
  main(int argc, char *argv[])
  {
    register int i, j, k, jl, jr, t, tl, tr;
    Graph *g;
    ⟨Process the command line 2*⟩;
    if (r ≡ 0) ⟨Handle the trivial case directly 6⟩
    else {
      ⟨Build the complete binary tree with n leaves 3⟩;
      for (i = n − 2; i; i−−) ⟨Generate the clauses for node i 4*⟩;
      ⟨Generate the clauses at the root 5⟩;
    }
  }
```

2*. ⟨Process the command line 2*⟩ ≡

```
  if (argc ≠ 5 ∨ sscanf(argv[1], "%d", &n) ≠ 1 ∨ sscanf(argv[2], "%d", &r) ≠ 1 ∨ sscanf(argv[4], "%d", &kk) ≠ 1)
    {
    fprintf(stderr, "Usage: %s n r foo.gb k\n", argv[0]);
    exit(−1);
  }
  g = restore_graph(argv[3]);
  if (¬g) {
    fprintf(stderr, "I can't input the graph '%s'!\n", argv[3]);
    exit(−2);
  }
  if (g→n ≠ n) fprintf(stderr, "Warning: The graph has %ld vertices, not %d!\n", g→n, n);
  if (n > nmax) {
    fprintf(stderr, "Recompile me: I'd don't allow n>%d\n", nmax);
    exit(−2);
  }
  if (r < 0 ∨ r ≥ n) {
    fprintf(stderr, "Eh? r should be between 0 and n-1!\n");
    exit(−2);
  }
  printf("~ sat-threshold-bb %d %d\n", n, r);
```

This code is used in section 1*.

3. The tree has $2n-1$ nodes, with 0 as the root; the leaves start at node $n-1$. Nonleaf node k has left child $2k+1$ and right child $2k+2$. Here we simply fill the *count* array.

⟨ Build the complete binary tree with n leaves 3 ⟩ ≡

```
for (k = n + n − 2; k ≥ n − 1; k−−) count[k] = 1;
for ( ; k ≥ 0; k−−) count[k] = count[k + k + 1] + count[k + k + 2];
if (count[0] ≠ n) fprintf(stderr, "I'm␣totally␣confused.\n");
```

This code is used in section 1*.

4* If there are t leaves below node i, we introduce $k = \min(r,t)$ variables Bi+1.j for $1 \le j \le k$. This variable is 1 if (but not only if) at least j of those leaf variables are true. If $t > r$, we also assert that no $r+1$ of those variables are true.

```
#define xbar(k) printf("~%s.%d", (g→vertices + (k) − n + 1)→name, kk)
```

⟨ Generate the clauses for node i 4* ⟩ ≡

```
{
  t = count[i], tl = count[i + i + 1], tr = count[i + i + 2];
  if (t > r + 1) t = r + 1;
  if (tl > r) tl = r;
  if (tr > r) tr = r;
  for (jl = 0; jl ≤ tl; jl++)
    for (jr = 0; jr ≤ tr; jr++)
      if ((jl + jr ≤ t) ∧ (jl + jr) > 0) {
        if (jl) {
          if (i + i + 1 ≥ n − 1) xbar(i + i + 1);
          else printf("~B%d.%d", i + i + 2, jl);
        }
        if (jr) {
          printf("␣");
          if (i + i + 2 ≥ n − 1) xbar(i + i + 2);
          else printf("~B%d.%d", i + i + 3, jr);
        }
        if (jl + jr ≤ r) printf("␣B%d.%d\n", i + 1, jl + jr);
        else printf("\n");
      }
}
```

This code is used in section 1*.

5. Finally, we assert that at most r of the x's are true, by implicitly asserting that the (nonexistent) variable `B1.`r+1 is false.

⟨ Generate the clauses at the root 5 ⟩ ≡

```
tl = count[1], tr = count[2];
if (tl > r) tl = r;
for (jl = 1; jl ≤ tl; jl++) {
  jr = r + 1 − jl;
  if (jr ≤ tr) {
    if (1 ≥ n − 1) xbar(1);
    else printf("~B2.%d", jl);
    printf("␣");
    if (2 ≥ n − 1) xbar(2);
    else printf("~B3.%d", jr);
    printf("\n");
  }
}
```

This code is used in section 1*.

6. ⟨ Handle the trivial case directly 6 ⟩ ≡

```
{
  for (i = 1; i ≤ n; i++) {
    xbar(n − 2 + i);
    printf("\n");
  }
}
```

This code is used in section 1*.

7* Index.

The following sections were changed by the change file: 1, 2, 4, 7.

argc: 1* 2*
argv: 1* 2*
count: 1* 3, 4* 5.
exit: 2*
fprintf: 2* 3.
Graph: 1*
i: 1*
j: 1*
jl: 1* 4* 5.
jr: 1* 4* 5.
k: 1*
kk: 1* 2* 4*
main: 1*
n: 1*
name: 4*
nmax: 1* 2*
printf: 2* 4* 5, 6.
r: 1*
restore_graph: 2*
sscanf: 2*
stderr: 2* 3.
t: 1*
tl: 1* 4* 5.
tr: 1* 4* 5.
vertices: 4*
xbar: 4* 5, 6.

⟨ Build the complete binary tree with n leaves 3 ⟩ Used in section 1*.
⟨ Generate the clauses at the root 5 ⟩ Used in section 1*.
⟨ Generate the clauses for node i 4* ⟩ Used in section 1*.
⟨ Handle the trivial case directly 6 ⟩ Used in section 1*.
⟨ Process the command line 2* ⟩ Used in section 1*.

SAT-THRESHOLD-BB-GRAPHS

	Section	Page
Intro	1	1
Index	7	4