1. Intro. Supplementary clauses to speed up `sat-color-order queen`n`x`n`.gb` d: These clauses say that every k-clique must contain at least one relatively high color and at least one relative low color.

Furthermore, since that idea worked so well, I'm trying to see if it can be used twice: I generate also a random permutation and maintain order variables for that order as well as the natural order.

```
#define maxd 100
#include <stdio.h>
#include <stdlib.h>
#include "gb_flip.h"
  int n;      /* this many queens */
  int d;      /* this many colors */
  int seed;     /* seed for gb_init_rand */
  int perm[maxd];     /* the random permutation */
  main(int argc,char *argv[])
  {
    register int i, j, k, l;
    ⟨ Process the command line 2 ⟩;
    ⟨ Set up the new permutation 3 ⟩;
    ⟨ Generate the clauses 4 ⟩;
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
  if (argc ≠ 4 ∨ sscanf(argv[1],"%d",&n) ≠ 1 ∨ sscanf(argv[2],"%d",&d) ≠ 1 ∨ sscanf(argv[3],"%d",
        &seed) ≠ 1) {
    fprintf(stderr,"Usage: %s n d seed\n",argv[0]);
    exit(-1);
  }
  if (d < n) {
    fprintf(stderr,"The number of colors (%d) must be at least %d!\n",d,n);
    exit(-2);
  }
  if (d > maxd) {
    fprintf(stderr,"The number of colors (%d) must be at most %d!\n",d,maxd);
    exit(-2);
  }
  gb_init_rand(seed);
  printf("~ sat-queens-color-order-cliques2 %d %d %d\n",n,d,seed);
```

This code is used in section 1.

3. ⟨ Set up the new permutation 3 ⟩ ≡

```
  for (i = 1; i < d; i++) {
    j = gb_unif_rand(i + 1);
    perm[i] = perm[j];
    perm[j] = i;
  }
  printf("~");
  for (i = 0; i < d; i++) printf(" %d",perm[i]);
  printf("\n");
```

This code is used in section 1.

4. ⟨ Generate the clauses 4 ⟩ ≡

```
⟨ Generate consistency clauses for perm 5 ⟩;
for (k = 0; k < n; k++) {
  ⟨ Generate cliques for row k 6 ⟩;
  ⟨ Generate cliques for column k 7 ⟩;
}
for (k = 1; k ≤ n + n − 3; k++) ⟨ Generate cliques for i + j = k 8 ⟩;
for (k = 2 − n; k ≤ n − 2; k++) ⟨ Generate cliques for i − j = k 9 ⟩;
```

This code is used in section 1.

5. ⟨ Generate consistency clauses for perm 5 ⟩ ≡

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++) {
    for (k = 0; k < d; k++) {
      if (k > 0 ∧ k < d − 1) printf("~%d.%d!%d␣%d.%d!%d\n", i, j, perm[k], i, j, perm[k + 1]);
      if (perm[k] + 1 < d) {
        if (k + 1 < d) printf("~%d.%d!%d", i, j, perm[k + 1]);
        if (k) printf("␣%d.%d!%d", i, j, perm[k]);
        printf("␣%d.%d<%d\n", i, j, perm[k] + 1);
      }
      if (perm[k]) {
        if (k + 1 < d) printf("~%d.%d!%d", i, j, perm[k + 1]);
        if (k) printf("␣%d.%d!%d", i, j, perm[k]);
        printf("␣~%d.%d<%d\n", i, j, perm[k]);
      }
      if (k + 1 < d) {
        if (perm[k] + 1 < d) printf("~%d.%d<%d", i, j, perm[k] + 1);
        if (perm[k]) printf("␣%d.%d<%d", i, j, perm[k]);
        printf("␣%d.%d!%d\n", i, j, perm[k + 1]);
      }
      if (k) {
        if (perm[k] + 1 < d) printf("~%d.%d<%d", i, j, perm[k] + 1);
        if (perm[k]) printf("␣%d.%d<%d", i, j, perm[k]);
        printf("␣~%d.%d!%d\n", i, j, perm[k]);
      }
    }
  }
```

This code is used in section 4.

6. ⟨ Generate cliques for row k 6 ⟩ ≡

```
{
  for (j = 0; j < n; j++) printf("␣%d.%d<%d", k, j, d − n + 1);
  printf("\n");
  for (j = 0; j < n; j++) printf("␣~%d.%d<%d", k, j, n − 1);
  printf("\n");
  for (j = 0; j < n; j++) printf("␣%d.%d!%d", k, j, perm[d − n + 1]);
  printf("\n");
  for (j = 0; j < n; j++) printf("␣~%d.%d!%d", k, j, perm[n − 1]);
  printf("\n");
}
```

This code is used in section 4.

7. ⟨ Generate cliques for column k 7 ⟩ ≡

```
{
  for (j = 0; j < n; j++) printf("␣%d.%d<%d", j, k, d - n + 1);
  printf("\n");
  for (j = 0; j < n; j++) printf("␣~%d.%d<%d", j, k, n - 1);
  printf("\n");
  for (j = 0; j < n; j++) printf("␣%d.%d!%d", j, k, perm[d - n + 1]);
  printf("\n");
  for (j = 0; j < n; j++) printf("␣~%d.%d!%d", j, k, perm[n - 1]);
  printf("\n");
}
```

This code is used in section 4.

8. ⟨ Generate cliques for $i + j = k$ 8 ⟩ ≡

```
{
  if (k < n) {
    l = k + 1;
    for (i = 0; i ≤ k; i++) printf("␣%d.%d<%d", i, k - i, d - l + 1);
    printf("\n");
    for (i = 0; i ≤ k; i++) printf("␣~%d.%d<%d", i, k - i, l - 1);
    printf("\n");
    for (i = 0; i ≤ k; i++) printf("␣%d.%d!%d", i, k - i, perm[d - l + 1]);
    printf("\n");
    for (i = 0; i ≤ k; i++) printf("␣~%d.%d!%d", i, k - i, perm[l - 1]);
    printf("\n");
  } else {
    l = n + n - 1 - k;
    for (i = n - l; i < n; i++) printf("␣%d.%d<%d", i, k - i, d - l + 1);
    printf("\n");
    for (i = n - l; i < n; i++) printf("␣~%d.%d<%d", i, k - i, l - 1);
    printf("\n");
    for (i = n - l; i < n; i++) printf("␣%d.%d!%d", i, k - i, perm[d - l + 1]);
    printf("\n");
    for (i = n - l; i < n; i++) printf("␣~%d.%d!%d", i, k - i, perm[l - 1]);
    printf("\n");
  }
}
```

This code is used in section 4.

9. ⟨ Generate cliques for $i-j=k$ 9 ⟩ ≡

```
{
  if (k > 0) {
    l = n - k;
    for (i = k; i < n; i++) printf("␣%d.%d<%d", i, i - k, d - l + 1);
    printf("\n");
    for (i = k; i < n; i++) printf("␣~%d.%d<%d", i, i - k, l - 1);
    printf("\n");
    for (i = k; i < n; i++) printf("␣%d.%d!%d", i, i - k, perm[d - l + 1]);
    printf("\n");
    for (i = k; i < n; i++) printf("␣~%d.%d!%d", i, i - k, perm[l - 1]);
    printf("\n");
  } else {
    l = n + k;
    for (i = 0; i < n + k; i++) printf("␣%d.%d<%d", i, i - k, d - l + 1);
    printf("\n");
    for (i = 0; i < n + k; i++) printf("␣~%d.%d<%d", i, i - k, l - 1);
    printf("\n");
    for (i = 0; i < n + k; i++) printf("␣%d.%d!%d", i, i - k, perm[d - l + 1]);
    printf("\n");
    for (i = 0; i < n + k; i++) printf("␣~%d.%d!%d", i, i - k, perm[l - 1]);
    printf("\n");
  }
}
```

This code is used in section 4.

10. Index.

SAT-QUEENS-COLOR-ORDER-CLIQUES2