Sudoku Players' Forums

THE REAL DISTRIBUTION OF MINIMAL PUZZLES

Goto page Previous 1, 2, 3 ... , 36, 37, 38 Next

newtopic postreply Sudoku Players' Forums Forum Index -> General/puzzle

View previous topic :: View next topic

Author	Message
denis_berthier Joined: 19 Jun 2007 Posts: 931 Location: Paris, France Back to top	Posted: Wed Oct 14, 2009 5:50 am Post subject: quote edit **Paul**, I'll have a more precise estimate of speed in the evening, but I can't resist the temptation to give you a first quick one now: 3.5 to 4 times faster than suexg-cb-optim48-U4. After the optim48 and the U4 optimisations, I didn't expect an improvement by such a factor. profile pm www
Red Ed Joined: 06 Jun 2005 Posts: 797	Posted: Wed Oct 14, 2009 6:22 am Post subject: quote **denis_berthier wrote:** **Red Ed wrote:** **denis_berthier wrote:** how many complete grids? how many 24-clue subgrids for each? One 24 per complete grid First possible source of bias: one 24-clue per complete grid. We know that the suexg algorithm is strongly biased because it produces 1 minimal per complete grid. Why choosing one 24 (not minimal) per complete grid would not introduce a bias? Second possible source of bias: the source of complete grids; they don't all have the same numbers of 24s and all their 24s don't have the same number of 31 minimals. It's one 24-clue subgrid per complete grid. I don't introduce any conditions on how that's chosen. Each solution grid has the same number of 24-clue subgrids, so that's not a source of bias. **Quote:** So, same question: how many complete grids? I already answered that: there's one complete grid per 24 and the number of 24s (yielding various numbers of 31s) is given <here>. Add 'em up: it's about 21300. **Quote:** For the estimate consistent with mine, it can be interesting for you, but it couldn't bring much to me now that I have more than a million minimals. You should be pleased to have a check: experimental results look better when corroborated. **Quote:** - does your estimate of the 30s relies on an estimate of the 24s? No. **Quote:** - what if you had to produce the full number-of-clues distribution? That's not the aim! You're burning CPU to do the main part of the distribution: my code is designed to attack the right tail. **Quote:** - how does it scale with precision? Square root rule: four times the work, half the standard error of the estimator.

Quote:

> **Red Ed wrote:**
>
> > well done for indicating the (informal) 1-sigma standard error in your recent post.
>
> That was your suggestion. When you are constructive, I can hear you.

The informal approach is less than wonderful for very low counts. I've been thinking about using significance intervals instead, which are - ahem - a little non-standard but have a very natural interpretation. Will post on that later.

Back to top profile pm

Red Ed

Joined: 06 Jun 2005
Posts: 797

Posted: Wed Oct 14, 2009 6:59 am Post subject: quote

btw - I notice you edited your post to note that 22K solution grids was "a very small number". True, it is; but don't be fooled into thinking that makes the estimated number of 31s especially unreliable -- my method consumes far fewer solution grids than yours.

On the parts of the distribution that we're each focussing on, it seems we each have a method tuned for the purpose: my supersets method has no hope of beating yours below 30 clues (nor does it try to); and your paths method has no hope of beating mine above 30 clues (nor do you proclaim any interest in doing). You could almost call it a team effort.

Back to top profile pm

denis_berthier

Joined: 19 Jun 2007
Posts: 931
Location: Paris, France

Posted: Wed Oct 14, 2009 7:19 am Post subject: quote edit

> **Red Ed wrote:**
>
> > **denis_berthier wrote:**
> >
> > > - what if you had to produce the full number-of-clues distribution?
> >
> > That's not the aim! You're burning CPU to do the main part of the distribution: my code is designed to attack the right tail.

That's what I thought. But then, how can you compare the speed of two programs that are not designed to do the same thing?

The work on the controlled-bias generator (in various forms and degrees: ideas or coding or providing grids or running the programs ...) has implied many persons in addition to me: eleven, Paul, gsf, Allan, Mike, Coloin, David, JPF - wish I'm forgetting no one!
Thanks to this collaboration, between the first and the last versions of the program, a factor ~ 200 in speed has been gained, which allowed to compute precise estimates for the distribution of clues and variables dependent on it.

> **Red Ed wrote:**
>
> > On the parts of the distribution that we're each focussing on, it seems we each have a method tuned for the purpose: my supersets method has no hope of beating yours below 30 clues (nor does it try to); and your paths method has no hope of beating mine above 30 clues (nor do you proclaim any interest in doing).

In a previous post, you suggested that you alone had done 750 times better than us all.
With your last post, truth is re-established, I didn't ask more.

> **Red Ed wrote:**
>
> > You could almost call it a team effort.

Almost. I've been used to better relations within a team.

Back to top profile pm www

Red Ed

Joined: 06 Jun 2005
Posts: 797

Posted: Wed Oct 14, 2009 7:24 am Post subject: quote

> **denis_berthier wrote:**
>
> > how can you compare the speed of two programs that are not designed to do the same thing?

Do you want me to explain the technical details? The aim of the full-scans-equivalent figures is to show how my method performs relative to yours in the right tail of the distribution. It's to make clear to the reader that your method is not the optimum one when used beyond its design parameters (nor should it be; that's no bad reflection on the authors).

> **denis_berthier wrote:**
>
> > In a previous post, you suggested that you alone had done 750 times better than us all.
> > With your last post, truth is re-established, I didn't ask more.

At what point have I lied about the performance of my code? Don't be silly, Denis. I've been careful all along to point out

that I'm addressing the right tail of the distribution - and, yes, in that tail I have done better than "you all". So what?

Back to top profile pm

denis_berthier

Posted: Wed Oct 14, 2009 7:33 am Post subject: quote edit

Joined: 19 Jun 2007
Posts: 931
Location: Paris, France

Red Ed,
My previous post says all that I had to say on this topic.

Back to top profile pm www

Red Ed

Posted: Wed Oct 14, 2009 7:37 am Post subject: quote

Joined: 06 Jun 2005
Posts: 797

No answer? Hardly surprising.

I'll just remind the reader of the "offending" quote - they can judge for themselves whether you're whining about nothing:

> **Red Ed wrote:**
>
> In other words, if I've got my stats right (done in a rush ...), for the purposes of estimating the number of proper minimal 31s per grid (and only that part of the distribution), **the supersets method runs at a rate equivalent to about 750 controlled-bias scans of gsf's collection per day**.

Back to top profile pm

PIsaacson

Posted: Wed Oct 14, 2009 8:01 am Post subject: quote

Joined: 02 Jul 2008
Posts: 218
Location: Campbell, CA

> **denis_berthier wrote:**
>
> **Paul**,
> I'll have a more precise estimate of speed in the evening, but I can't resist the temptation to give you a first quick one now: 3.5 to 4 times faster than suexg-cb-optim48-U4.
> After the optim48 and the U4 optimisations, I didn't expect an improvement by such a factor.

It's not often that you can get such a massive speed up with such little effort. I'm still profiling the code to see if there are any other potential low hanging fruit, but so far nothing looks promising for the controlled bias phase. I'm hopeful that Eleven will review the changes and release an "official" BSF (best so far) version for all to use.

However, for the internal grid generation phase, there is the potential for implementing the Edith Ahola 159 algorithm to really speed up that section of the code. With the ability to pipe/steam in grids, the internal grid generator is less important, but I'm working on replacing it with the EA159 algorithm anyway. As a standalone program, it produces almost 100k puzzles/second on my AMD 4200+, which is an old and not exceptionally fast cpu.

I'm still catching up on this thread, but one thing I'd like to experiment with is bottom-up grid generator that uses the bb_solver code. I believe I read somewhere that bottom-up types tend to generate puzzles with slightly higher average SE/NRCZT ratings. Is it worth investigating a controlled bias bottom-up generator? Did I miss out on discussions regarding the pros/cons of this?

Cheers,
Paul

Back to top profile pm

denis_berthier

Posted: Wed Oct 14, 2009 8:52 am Post subject: quote edit

Joined: 19 Jun 2007
Posts: 931
Location: Paris, France

> **PIsaacson wrote:**
>
> I'm still catching up on this thread, but one thing I'd like to experiment with is bottom-up grid generator that uses the bb_solver code. I believe I read somewhere that bottom-up types tend to generate puzzles with slightly higher average SE/NRCZT ratings. Is it worth investigating a controlled bias bottom-up generator? Did I miss out on discussions regarding the pros/cons of this?

bottom-up generators tend to generate puzzles with still *fewer clues* and *lower* average SE/NRCZT ratings than top-down.
The #clues bias is so large that building controlled-bias generators from them seems hopeless.
You can find all the related results on my web pages.

Having a *fast and unbiased* complete grids generator could be useful. Not everyone wants to generate the full gsf collection.

Back to top profile pm www

eleven

Posted: Wed Oct 14, 2009 10:46 am Post subject: quote

Joined: 10 Feb 2008

Exciting news, unfortunately i have little time now. But i did 2 things.

Posts: 532

Paul,
well done, i had a quick look at the bb version.
I compared the speed for a 10 mio suexg grids sample and it was 3 times faster !
I also tried it without the U4 test and it was 8% faster for this sample with avg 10.6 U4s, so i think, it is better now to comment out the U4 test:

Code:

```
in main()
//   calc4unavoid();
and in solve()
//   if (part && empty4unavoid())
//     return 2;
```

Red Ed,
after you wrote, that your method would be similar to, what i posted, i fixed a bug (now about 75% of the 24's can be excluded at the beginning) and put 7 loops around, where
- a number is placed in the next free cell
- a singles test is done, if it is not miminal (given and placed clues only)
- singles of the current puzzle are added (need not be tested)

It seems to work, but is 20 times slower than your version ! So i am curious, what your trick is.

The result for 1000 random 24's in suexg grids (took 7.5 hours !) was 103 29's, 32 30's and 21 31's. Interesting was, that a single 24 had 62 29's, 8 30's and 2 31's. So these high clues are strongly clustered.

Back to top profile pm

Red Ed

Posted: Wed Oct 14, 2009 5:45 pm Post subject: quote

eleven - code to follow, but in brief there are a few related tricks:

Joined: 06 Jun 2005
Posts: 797

1. Always being aware of the implied clues, and obviously never adding one of them; whenever calling the solver, always make use of preprocessing done on a smaller number of clues.
2. Making sure that all n(n-1)/2 pairs of clues amongst n (up to 7) clues added satisfy safe[clue1][clue2]==1 where safe[][] is a pretabulated array of which clue pairs can be added to the base 24 grid without introducing redundancy.
3. At 26 clues, build a very compact (bitmasked) list of *all* solutions; and as each clue is added, filter that list to make it smaller. At 30 (or 31 or whatever target number of) clues, confirm that the puzzle is proper by simply checking that the solution list has length 1. (Of course checking minimality still requires calls to the solver - but those can use preprocessing as per point 1 above.)

On a single 1.4GHz CPU, this appears to be equivalent from the p.o.v. of precision* to about **70 full scans of gsf's collection per day**. Remember though that this is the figure for estimation only of the number of proper minimal 30s; and the code is not necessarily bug-free yet!

*: this is a comparison based on the standard error of the estimator. The supersets estimator - like the subsets estimator before it - has lower variance and consumes fewer solution grids than suexg-cb path estimator, but it's much more expensive to compute. It comes into its own for 31+ (or maybe 30+) clue minimals.

Back to top profile pm

Red Ed

Posted: Wed Oct 14, 2009 6:47 pm Post subject: quote

And - without waiting to tidy it up properly - here's the code.

Joined: 06 Jun 2005
Posts: 797

You need the mtwist package to compile it and a grids generator to pipe into it, example: **./generator | ./this_code**

Output is not especially nice to read, there are no command line parameters, the editable #defines are not marked as such and there is no internal sanity-checking of them, the code itself is untidy and it quite possibly contains bugs. You have been warned.

But at least with the code in front of you, you can get a feel for its performance.

Code:

```
/* 31a_posted.c
 * -- the still probably a bit flakey version posted to the Players' Forum
 */

/* Compilation:
 * \mingw\bin\gcc -o 31a 31a.c mtwist-0.6\mtwist.c mtwist-0.6\randistrs.c -Wall -O9
 */

#include "mtwist-0.6/mtwist.h"
#include "mtwist-0.6/randistrs.h"
```

```
#include <stdio.h>
#include <stdlib.h>
#include <mem.h>
#include <math.h>
#include <time.h>

#if DEBUG
#define SUBSZ (22)
#define TARGET (30)
#else
#define SUBSZ (24)
#define TARGET (30)
#endif
#define NTMPL (46656)

#define SAVE_N(D,L) old_n##D[L] = n##D;

#define COMPRESS(D,L)                                          \
  {                                                            \
    int x;                                                     \
    n##D = old_n##D[L];                                        \
    for (x=0; x<n##D; x++) {                                   \
      if (  (mask[0]&active.t[D][x][0])                        \
          || (mask[1]&active.t[D][x][1])                       \
          || (mask[2]&active.t[D][x][2]) ) {                   \
        uint32 tmp           = active.t[D][x][0];            \
        active.t[D][x][0]    = active.t[D][--n##D][0];       \
        active.t[D][n##D][0] = tmp;                          \
        tmp                  = active.t[D][x][1];            \
        active.t[D][x][1]    = active.t[D][n##D][1];         \
        active.t[D][n##D][1] = tmp;                          \
        tmp                  = active.t[D][x][2];            \
        active.t[D][x--][2]  = active.t[D][n##D][2];         \
        active.t[D][n##D][2] = tmp;                          \
      }                                                      \
    }                                                        \
  }

#define START_LOOP(D)                                                          \
  for (i##D=0; i##D<n##D; i##D++) {                                            \
    uint32 t[3];                                                               \
    t[0] = active.t[D][i##D][0];                                               \
    t[1] = active.t[D][i##D][1];                                               \
    t[2] = active.t[D][i##D][2];                                               \
    if ( 0==(t[0]&mask[0]) && 0==(t[1]&mask[1]) && 0==(t[2]&mask[2]) ) {       \
      mask[0] ^= t[0];                                                         \
      mask[1] ^= t[1];                                                         \
      mask[2] ^= t[2];

#define END_LOOP         \
      mask[0] ^= t[0];   \
      mask[1] ^= t[1];   \
      mask[2] ^= t[2];   \
    }                    \
  }

typedef unsigned int uint32;

typedef struct {
  int    notlast;
  uint32 t[10][NTMPL][3];
  uint32 nt[10];
  int    idx2digit[10];
} tlist_t;

static int   slist_mem = 0, slist_len[1+TARGET-SUBSZ];
static uint32 (*slist)[3] = NULL;

static tlist_t all_tmpl;
static tlist_t active;
static tlist_t base_tmpl;
static tlist_t punctured[81];

static int jct[1001] = {0};

/* ------------------------------------------------------
 * find all 46656 unconstrained templates
 * ------------------------------------------------------
 */

static void get_unconstrained_templates(int box) {
  static int grid[9][9];
  int i0, j0;
  int i, j, k;

  if (box == 0) {
    for (i=0; i<81; i++) grid[i/9][i%9] = 0;
    all_tmpl.nt[1] = 0;
  }
```

```
  if (box == 9) {
    uint32 t[3] = {0};
    if (all_tmpl.nt[1] == NTMPL) {
      fprintf(stderr,"template overflow!\n");
      exit(1);
    }
    for (i=0; i<81; i++) if (grid[i/9][i%9]) t[i/32] |= 1<<(i%32);
    all_tmpl.t[1][all_tmpl.nt[1]][0] = t[0];
    all_tmpl.t[1][all_tmpl.nt[1]][1] = t[1];
    all_tmpl.t[1][all_tmpl.nt[1]][2] = t[2];
    all_tmpl.nt[1]++;
    return;
  }

  i0 = 3 * (box/3);
  j0 = 3 * (box%3);
  for (i=i0; i<i0+3; i++) {
    for (j=j0; j<j0+3; j++) {
      for (k=0; k<9; k++) if (grid[k][j] == 1) break; if (k<9) continue;
      for (k=0; k<9; k++) if (grid[i][k] == 1) break; if (k<9) continue;
      grid[i][j] = 1;
      get_unconstrained_templates(box+1);
      grid[i][j] = 0;
    }
  }

  if (box) return;

  if (all_tmpl.nt[1] != NTMPL) {
    fprintf(stderr,"template underflow! (all_tmpl.nt[1]=%d)\n",all_tmpl.nt[1]);
    exit(1);
  }

  for (i=2; i<=9; i++) {
    for (j=0; j<NTMPL; j++) {
      all_tmpl.t[i][j][0] = all_tmpl.t[1][j][0];
      all_tmpl.t[i][j][1] = all_tmpl.t[1][j][1];
      all_tmpl.t[i][j][2] = all_tmpl.t[1][j][2];
    }
    all_tmpl.nt[i] = NTMPL;
  }

  for (i=1; i<=9; i++) all_tmpl.idx2digit[i] = i;
  all_tmpl.notlast = 0;

  return;
}

/* -------------------------------------------------------
 * filter templates for a given puzzle
 * - if hole is specified then introduce a puncture
 * -------------------------------------------------------
 */

static void filter_templates(tlist_t *src, tlist_t *dst, int (*puz)[9], int hole) {
  int pass, order[10];

  dst->notlast = src->notlast;
  if (0<=hole && hole<81) {
    if (src->notlast) {
      fprintf(stderr,"PANIC: attempt to double-puncture\n");
      exit(-1);
    }
    dst->notlast = puz[hole/9][hole%9];
    if (!dst->notlast) {
      fprintf(stderr,"PANIC: attempt to puncture with a blank\n");
      exit(-1);
    }
  }

  for (pass=0; pass<2; pass++) {
    int dstidx;

    for (dstidx=1; dstidx<=9; dstidx++) {
      uint32 hit[3]={0}, miss[3]={0};
      int    srcidx   = pass ? order[dstidx] : dstidx;
      int    srcdigit = src->idx2digit[srcidx];
      int    i;

      for (i=0; i<81; i++) {
        int zap = (i==hole && srcdigit==puz[i/9][i%9]);
        if (!zap && puz[i/9][i%9]==srcdigit) {
          hit[i/32] |= 1u << (i%32);
        } else if (zap || (i!=hole && puz[i/9][i%9])) {
          miss[i/32] |= 1u << (i%32);
        }
      }
      dst->nt[dstidx] = 0;
```

```
      for (i=0; i<src->nt[srcidx]; i++) {
        if (  (miss[0]&src->t[srcidx][i][0])==0 && (hit[0]&~src->t[srcidx][i][0])==0 \
           && (miss[1]&src->t[srcidx][i][1])==0 && (hit[1]&~src->t[srcidx][i][1])==0 \
           && (miss[2]&src->t[srcidx][i][2])==0 && (hit[2]&~src->t[srcidx][i][2])==0 ) {
          if (pass == 1) {
            dst->t[dstidx][dst->nt[dstidx]][0] = src->t[srcidx][i][0];
            dst->t[dstidx][dst->nt[dstidx]][1] = src->t[srcidx][i][1];
            dst->t[dstidx][dst->nt[dstidx]][2] = src->t[srcidx][i][2];
          }
          dst->nt[dstidx]++;
        }
      }
    }

    if (pass == 0) {
      for (dstidx=1; dstidx<=9; dstidx++) {
        int mind=0, d;
        for (d=1; d<=9; d++) if (mind==0 || dst->nt[d]<dst->nt[mind]) mind=d;
        order[dstidx] = mind;
        dst->nt[mind] = NTMPL+1;
      }
      if (src->idx2digit[order[9]] == dst->notlast) {
        int tmp = order[9];
        order[9] = order[8];
        order[8] = tmp;
      }
    }
  }

  for (pass=1; pass<=9; pass++) dst->idx2digit[pass] = src->idx2digit[order[pass]];

  return;
}

/* ----------------------------------------------------
 * copy templates
 * ----------------------------------------------------
 */

static void copy_templates(tlist_t *src, tlist_t *dst) {
  int i, j;

  dst->notlast = src->notlast;
  for (i=1; i<=9; i++) {
    for (j=0; j<src->nt[i]; j++) {
      dst->t[i][j][0] = src->t[i][j][0];
      dst->t[i][j][1] = src->t[i][j][1];
      dst->t[i][j][2] = src->t[i][j][2];
    }
    dst->nt[i] = src->nt[i];
    dst->idx2digit[i] = src->idx2digit[i];
  }

  return;
}

/* ----------------------------------------------------
 * count
 * ----------------------------------------------------
 */

static int count(tlist_t *tlp, int maxct, int (*solution)[9], int (*base)[9]) {
  static uint32 mask[3], soltmpl[10][3];
  int    n1, n2, n3, n4, n5, n6, n7, n8;
  int    old_n3[2];
  int    old_n4[3];
  int    old_n5[4];
  int    old_n6[5];
  int    old_n7[6];
  int    old_n8[7];
  int    i1, i2, i3, i4, i5, i6, i7, i8;
  int    ct;

  if (tlp != &active) copy_templates(tlp,&active);

  if (maxct < 0) {
    int i, j;
    if (maxct!=-1 || !solution || !base) {
      fprintf(stderr,"bad parameters to count()\n");
      exit(-1);
    }
    for (i=1; i<=9; i++) {
      soltmpl[i][0] = soltmpl[i][1] = soltmpl[i][2] = 0;
      for (j=0; j<81; j++) if (solution[j/9][j%9] == active.idx2digit[i]) soltmpl[i][j/32] |=
1<<(j%32);
    }
  }

  n1 = active.nt[1];
```

```
  n2 = active.nt[2];
  n3 = active.nt[3];
  n4 = active.nt[4];
  n5 = active.nt[5];
  n6 = active.nt[6];
  n7 = active.nt[7];
  n8 = active.nt[8];

  SAVE_N(3,1)
  SAVE_N(4,1)
  SAVE_N(5,1)
  SAVE_N(6,1)
  SAVE_N(7,1)
  SAVE_N(8,1)
  for (i1=ct=0; i1<n1; i1++) {
    mask[0] = active.t[1][i1][0];
    mask[1] = active.t[1][i1][1];
    mask[2] = active.t[1][i1][2];
    COMPRESS(3,1)
    COMPRESS(4,1) SAVE_N(4,2)
    COMPRESS(5,1) SAVE_N(5,2)
    COMPRESS(6,1) SAVE_N(6,2)
    COMPRESS(7,1) SAVE_N(7,2)
    COMPRESS(8,1) SAVE_N(8,2)
    START_LOOP(2)
      COMPRESS(4,2)
      COMPRESS(5,2) SAVE_N(5,3)
      COMPRESS(6,2) SAVE_N(6,3)
      COMPRESS(7,2) SAVE_N(7,3)
      COMPRESS(8,2) SAVE_N(8,3)
      START_LOOP(3)
        COMPRESS(5,3)
        COMPRESS(6,3) SAVE_N(6,4)
        COMPRESS(7,3) SAVE_N(7,4)
        COMPRESS(8,3) SAVE_N(8,4)
        START_LOOP(4)
          COMPRESS(6,4)
          COMPRESS(7,4) SAVE_N(7,5)
          COMPRESS(8,4) SAVE_N(8,5)
          START_LOOP(5)
            COMPRESS(7,5)
            COMPRESS(8,5) SAVE_N(8,6)
            START_LOOP(6)
              COMPRESS(8,6)
              START_LOOP(7)
                for (i8=0; i8<n8; i8++) {
                  if (  0 == (active.t[8][i8][0]&mask[0]) \
                     && 0 == (active.t[8][i8][1]&mask[1]) \
                     && 0 == (active.t[8][i8][2]&mask[2]) ) {
                    if (maxct == -1) {
                      uint32 soldigits[3] = {0};
                      uint32 nines[3] = {~0,~0,~0};

                      /* this part for the solutions list
                       */
#define GETSOL(D)                                          \
  soldigits[0] |= soltmpl[D][0] & active.t[D][i##D][0];  \
  soldigits[1] |= soltmpl[D][1] & active.t[D][i##D][1];  \
  soldigits[2] |= soltmpl[D][2] & active.t[D][i##D][2];
                      GETSOL(1)
                      GETSOL(2)
                      GETSOL(3)
                      GETSOL(4)
                      GETSOL(5)
                      GETSOL(6)
                      GETSOL(7)
                      GETSOL(8)
#define ZAPSOL9(D)                    \
  nines[0] &= ~active.t[D][i##D][0];  \
  nines[1] &= ~active.t[D][i##D][1];  \
  nines[2] &= ~active.t[D][i##D][2];
                      ZAPSOL9(1)
                      ZAPSOL9(2)
                      ZAPSOL9(3)
                      ZAPSOL9(4)
                      ZAPSOL9(5)
                      ZAPSOL9(6)
                      ZAPSOL9(7)
                      ZAPSOL9(8)
                      soldigits[0] |= soltmpl[9][0] & nines[0];
                      soldigits[1] |= soltmpl[9][1] & nines[1];
                      soldigits[2] |= soltmpl[9][2] & nines[2];
                      if (slist_mem == ct) {
                        slist = realloc(slist,3*sizeof(uint32)*++slist_mem);
                        if (!slist) {
                          perror("slist");
                          exit(-1);
                        }
                      }
```

```
                    slist[ct][0] = soldigits[0];
                    slist[ct][1] = soldigits[1];
                    slist[ct][2] = soldigits[2];
                  }
                  if (++ct>maxct && maxct>=0) return ct;
                }
              }
            END_LOOP /* 7 */
          END_LOOP /* 6 */
        END_LOOP /* 5 */
      END_LOOP /* 4 */
    END_LOOP /* 3 */
  END_LOOP /* 2 */
}  /* i2 */

  return ct;
}

/* ------------------------------------------------------
 * is it safe to add a single clue?
 * ------------------------------------------------------
 */

static int fast_ok_to_add1(int (*base)[9], int idx, int value) {
  int i, j, k;

  for (i=0; i<81; i++) if (base[i/9][i%9]) {
    for (j=1; j<=9; j++) {
      int digit = punctured[i].idx2digit[j];
      for (k=active.nt[j]=0; k<punctured[i].nt[j]; k++) {
        if ( (digit==value) == ((punctured[i].t[j][k][idx/32]&(1<<(idx%32)))!=0) ) {
          active.t[j][active.nt[j]][0] = punctured[i].t[j][k][0];
          active.t[j][active.nt[j]][1] = punctured[i].t[j][k][1];
          active.t[j][active.nt[j]][2] = punctured[i].t[j][k][2];
          active.nt[j]++;
        }
      }
    }
    if (count(&active,0,NULL,NULL) == 0) return 0;
  }

  return 1;
}

/* ------------------------------------------------------
 * is it safe to add a pair of clues?
 * ------------------------------------------------------
 */

static int fast_ok_to_add2(int (*base)[9], int idx1, int idx2, int value1, int value2) {
  int i, j;

  base[idx1/9][idx1%9] = value1;
  base[idx2/9][idx2%9] = value2;
  for (i=0; i<81; i++) if (base[i/9][i%9]) {
    j = base[i/9][i%9];
    base[i/9][i%9] = 0;
    filter_templates(&punctured[i],&active,base,999);
    base[i/9][i%9] = j;
    if (count(&active,0,NULL,NULL) == 0) break;
  }
  base[idx1/9][idx1%9] = 0;
  base[idx2/9][idx2%9] = 0;

  return (i == 81);
}

/* ------------------------------------------------------
 * do those TARGET-clue solves
 * ------------------------------------------------------
 */

static int do_jobs(int clues, int ptr, int (*safe2)[81], int (*base)[9], int (*solution)[9]) {
  static int history[TARGET-SUBSZ];
  uint32 intersection[3];
  int jackpots = 0;
  int i, j, k;
static int nr_tgts=0, nr_proper=0, nr_minchecks=0;

  if (clues == SUBSZ+2) {
    i = history[0]; base[i/9][i%9] = solution[i/9][i%9];
    i = history[1]; base[i/9][i%9] = solution[i/9][i%9];
    filter_templates(&base_tmpl,&active,base,999);
    i = history[0]; base[i/9][i%9] = 0;
    i = history[1]; base[i/9][i%9] = 0;

    slist_len[clues-SUBSZ] = count(&active,-1,solution,base);
#if 0
    printf("count(%2d,%2d) = ",history[0],history[1]); fflush(stdout);
```

```
    printf("%6d\n",slist_len[clues-SUBSZ]);
#endif
    intersection[0] = intersection[1] = intersection[2] = ~0;
    for (i=0; i<slist_len[clues-SUBSZ]; i++) {
      intersection[0] &= slist[i][0];
      intersection[1] &= slist[i][1];
      intersection[2] &= slist[i][2];
    }
  }

  if (clues > SUBSZ+2) {      /* now check the slist */
    uint32 basemask[3]={0};
    intersection[0] = intersection[1] = intersection[2] = ~0;
    for (i=0; i<81; i++) if (base[i/9][i%9]) basemask[i/32] |= 1<<(i%32);
    for (i=0; i<clues-SUBSZ; i++) basemask[history[i]/32] |= 1<<(history[i]%32);
#define SLOK(Z) (                   \
    !(basemask[0]&~slist[Z][0]) && \
    !(basemask[1]&~slist[Z][1]) && \
    !(basemask[2]&~slist[Z][2])    \
  )
    i = -1; j = slist_len[clues-SUBSZ-1]; k = 0;
    do {
      uint32 tmp;
      while (++i<j && SLOK(i)) {
        intersection[0] &= slist[i][0];
        intersection[1] &= slist[i][1];
        intersection[2] &= slist[i][2];
        k++;
      }
      while (i<--j && !SLOK(j)) ;
      if (i < j) {
        tmp = slist[i][0]; slist[i][0] = slist[j][0]; slist[j][0] = tmp;
        tmp = slist[i][1]; slist[i][1] = slist[j][1]; slist[j][1] = tmp;
        tmp = slist[i][2]; slist[i][2] = slist[j][2]; slist[j][2] = tmp;
        intersection[0] &= slist[i][0];
        intersection[1] &= slist[i][1];
        intersection[2] &= slist[i][2];
        k++;
      }
    } while (i < j);
    slist_len[clues-SUBSZ] = k;
  }

  if (clues == TARGET) {

    /*** TARGET clues ***/

nr_tgts++;
    for (i=0; i<TARGET-SUBSZ; i++) {
      j = history[i];
      base[j/9][j%9] = solution[j/9][j%9];
    }
    if (slist_len[clues-SUBSZ] == 1) {
nr_proper++;

      /* finally got a TARGET-clue subgrid with one solution ... now check minimality */

      for (i=0; i<81; i++) if (base[i/9][i%9]) {
        base[i/9][i%9] = 0;
        filter_templates(&punctured[i],&active,base,999);
        base[i/9][i%9] = solution[i/9][i%9];
nr_minchecks++;
        if (count(&active,0,NULL,NULL) == 0) break;
      }
      if (i == 81) {
        printf(">>> ");
        for (i=0; i<81; i++) printf("%d",base[i/9][i%9]);
        printf("\n");
        jackpots = 1;
      }
    }
    for (i=0; i<TARGET-SUBSZ; i++) {
      j = history[i];
      base[j/9][j%9] = 0;
    }
    return jackpots;

  } else {

    /*** fewer than TARGET clues ***/

    int good[81] = {0};
    for (j=ptr; j<81; j++) {
      if ( clues>=SUBSZ+2 && (intersection[j/32]&(1<<(j%32))) ) continue;
      for (i=0; i<clues-SUBSZ; i++) if (!safe2[history[i]][j]) break;
      if (i < clues-SUBSZ) continue;
      good[j] = 1;
    }
    for (i=80,j=0; i>=ptr; i--) if (good[i] && ++j<TARGET-clues) good[i] = 0;
```

```
    for (; ptr<81; ptr++) if (good[ptr]) {
      history[clues-SUBSZ] = ptr;
      jackpots += do_jobs(clues+1,ptr+1,safe2,base,solution);
    }

  }

if (clues == SUBSZ) {
  printf("Number of %2ds encountered: %d\n",TARGET,nr_tgts);
  printf("  Number that were proper: %d\n",nr_proper);
  printf("     Minchecks per proper: %f\n",nr_minchecks*1.0/nr_proper);
  nr_tgts = nr_proper = nr_minchecks = 0;
}

  return jackpots;
}

/* ------------------------------------------------------
 * count the number of TARGET-clue solves required
 * ------------------------------------------------------
 */

static int count_jobs(int clues, int ptr, int (*safe2)[81]) {
  static int history[TARGET-SUBSZ];
  int count, i;

  if (clues == TARGET) return 1;

  for (count=0; ptr<81; ptr++) {
    for (i=0; i<clues-SUBSZ; i++) if (!safe2[history[i]][ptr]) break;
    if (i < clues-SUBSZ) continue;
    history[i] = ptr;
    count += count_jobs(clues+1,ptr+1,safe2);
  }

  return count;
}

/* ------------------------------------------------------
 * PROGRAM START
 * ------------------------------------------------------
 */

int main(int argc, char **argv) {
  static int  solution[9][9], safe1[81], safe2[81][81];
  mt_state state;
  char line[100];
  int trials = 0;
  int  i, j, k;

  mts_goodseed(&state);

  get_unconstrained_templates(0);
  fprintf(stderr,"Go!\n");
  srand(time(NULL));
  while (1 == scanf("%99s",line)) {
    int base[9][9];

    jct[0]++;
    for (i=0; i<81; i++) base[i/9][i%9] = solution[i/9][i%9] = line[i] - '0';

    /* generate SUBSZ clue subgrid that has multiple solutions
     */
    printf("\n%d-clue subgrid #%d:\n",SUBSZ,++trials);
    for (i=SUBSZ; i<81; i++) {
      int pos;
      do {
        pos = rds_iuniform(&state,0,81);
      } while (base[pos/9][pos%9] == 0);
      base[pos/9][pos%9] = 0;
    }
#if DEBUG
{
char *s="123578694874691523965234781216749835439856172587123946341967258692385417758412369";
/*     t="000078000004601500005000080200700800409006000000003046001000250602000410058012309";
*/
/*4,5,11,12,14,15*/
char *b="000000000000000000005000080200700800409006000000000460010002506020004100580103090";
for(i=0;i<81;i++){
base[i/9][i%9]=b[i]-'0';
solution[i/9][i%9]=s[i]-'0';
}
}
#endif
    for (i=0; i<81; i++) printf("%d",base[i/9][i%9]); printf("\n");
    for (i=0; i<81; i++) printf("%d",solution[i/9][i%9]); printf("\n");

printf("^123456789^123456789^123456789^123456789^123456789^123456789^123456789^123456789^\n");
    filter_templates(&all_tmpl,&base_tmpl,base,999);
```

```
    if (count(&base_tmpl,1,NULL,NULL) == 1) {
      printf("Oops - unique solution\n");
      continue;
    }

    /* build template lists for SUBSZ-1 clue "punctured" subgrids of that,
     * checking for minimality of the SUBSZ clue grid as you do so
     */
    for (i=0; i<81; i++) if (base[i/9][i%9]) {
      filter_templates(&all_tmpl,&punctured[i],base,i);
      if (count(&punctured[i],0,NULL,NULL) == 0) {
        printf("Oops - redundant clues\n");
        goto MAINLOOP;
      }
    }

    /* TO DO, maybe: reduce punctured lists to templates where bit 95 == 1 */
    /* TO DO, maybe: reduce punctured lists to templates where bit 95 == 1 */
    /* TO DO, maybe: reduce punctured lists to templates where bit 95 == 1 */

    /* all blanks are safe singles initially
     */
    for (i=0; i<81; i++) safe1[i] = (base[i/9][i%9] == 0);

    /* implied clues are unsafe
     */
    for (i=0; i<81; i++) if (base[i/9][i%9] == 0) {
      base[i/9][i%9] = solution[i/9][i%9];
      filter_templates(&all_tmpl,&punctured[i],base,i);
      base[i/9][i%9] = 0;
      copy_templates(&punctured[i],&active);
      if (count(&active,0,NULL,NULL) == 0) { printf("Implied clue: %2d\n",i); safe1[i] = 0; }
    }

    /* new clues that imply existing ones are unsafe
     */
    for (i=j=0; i<81; i++) if (safe1[i]) {
      safe1[i] = fast_ok_to_add1(base,i,solution[i/9][i%9]);
      j += safe1[i];
    }
    printf("  Number of safe clues: %d\n",j);
    printf("             => at most %d safe pairs\n",(j*(j-1))/2);

    /* find which pairs of clues are safe to add
     */
    for (i=k=0; i<81; i++) for (j=i+1; j<81; j++) {
      if (safe1[i] && safe1[j]) {
        safe2[i][j] = fast_ok_to_add2(base,i,j,solution[i/9][i%9],solution[j/9][j%9]);
      } else {
        safe2[i][j] = 0;
      }
      k += safe2[i][j];
    }
    printf("  Number of safe pairs: %d\n",k);

    /* warn the viewer how many TARGET-clue supersets we'll be solving
     */
    printf("Number of %ds to solve: %d\n",TARGET,count_jobs(SUBSZ,0,safe2));

    /* solve!
     */
    j = do_jobs(SUBSZ,0,safe2,base,solution);
    if (j) {
      jct[0]--;
      if (j > 999) j = 1000;
      jct[j]++;
    }
MAINLOOP:
    for (j=0; j<1000; j++) if (jct[j]) printf("jct[%4d ] = %6d\n",j,jct[j]);
    printf("jct[1000+] = %6d\n",jct[1000]);
  }

  return 0;
}
```

Back to top profile pm

Red Ed

Posted: Wed Oct 14, 2009 6:48 pm Post subject: quote

PS: bug reports would be most welcome!

Joined: 06 Jun 2005
Posts: 797

Back to top profile pm

denis_berthier

Posted: Thu Oct 15, 2009 3:53 am Post subject:

quote edit

Joined: 19 Jun 2007
Posts: 931
Location: Paris, France

Paul, eleven ,
After a full day, it appears that the new suexg-cb is 3.3 times faster. Combined with gsf input, it now allows 2 full scans of his collection per day.
This is without the generation phase and before I saw eleven's U4 post. I'll now comment out the U4 test.

Back to top profile pm www

denis_berthier

Posted: Thu Oct 15, 2009 4:53 am Post subject:

quote edit

Joined: 19 Jun 2007
Posts: 931
Location: Paris, France

NEW RESULTS WITH SUEXG-CB, WITH GSF FILES AS INPUT

I now have 1,380,962 minimal puzzles, corresponding to 67 full scans of gsf's collection (obtained before Paul's optimisation).
The precision of the distribution is a little better.
The only new (unexpected) thing is the presence of a 32.
I let the generator run a little more, not that I'm expecting anything new, but mainly because it is now so fast, after Paul's last optimisation, that just seeing it run is a pleasure.
Also, as I begin to have a respectable number of 31s, I want to compute their SER and NRCZT to extend the range of their variations with the number of clues.

Code:

```
#clues  #instances       %               unbiased %      precision of unbiased %
        in cb sample     in cb sample    (estimated)     (estimated)
19      0                0.0             0.0
20      0                0.0             0.0
21      41               0.00297         3.38e-05        0.53e-05
22      1526             0.111           0.00343         8.78e-05
23      25884            1.874           0.149           0.00093
24      163694           11.85           2.280           0.0056
25      422451           30.59           13.415          0.021
26      467047           33.82           31.944          0.047
27      234963           17.01           32.736          0.068
28      57615            4.172           15.481          0.065
29      7243             0.524           3.557           0.042
30      481              0.0348          0.409           0.019
31      16               0.00116         0.0224          0.0056
32      1                7.24e-05        0.00219         0.00219


controlled-bias mean = 25.667     controlled-bias standard-deviation= 1.116
real mean = 26.577               real standard-deviation= 1.116
```

Figures (mean and sd) for the SER and NRCZT are unchanged.

Back to top profile pm www

Display posts from previous: All Posts Oldest First Go

newtopic postreply **Sudoku Players' Forums Forum Index -> General/puzzle**

All times are GMT
Goto page Previous 1, 2, 3 ... , 36, 37, 38 Next

Page 37 of 38

Stop watching this topic

Jump to: General/puzzle Go

You **can** post new topics in this forum
You **can** reply to topics in this forum
You **can** edit your posts in this forum
You **can** delete your posts in this forum
You **can** vote in polls in this forum