Sudoku Players' Forums

THE REAL DISTRIBUTION OF MINIMAL PUZZLES

Goto page Previous 1, 2, 3 ... , 24, 25, 26 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: 861 Location: Paris, France	Posted: Tue Sep 29, 2009 8:39 am Post subject: quote edit Then, I must have missed something. Using the command line: sudoku -e '(%#An)>1' -f'%v # %#An' 300-416.sudz shouldn't I find all the grids that have more than 1 automorphism, i.e. a non trivial one? (By trivial, I mean identity). But I find only 5203. Using sudoku -e '(%#An)>0' -f'%v # %#An' 300-416.sudz, I find all the 2097068 grids in the sample. Also, searching manually the expanded file, I find only 1 with 108 automorph and none with more. Edit: You were speaking of the whole file with 5 billion grids! I was speaking only of my little piece of it. If this is confirmed, then we don't have to care much for automorphisms in statistics. Last edited by denis_berthier on Tue Sep 29, 2009 9:45 am; edited 1 time in total
Back to top	profile pm www
denis_berthier Joined: 19 Jun 2007 Posts: 861 Location: Paris, France	Posted: Tue Sep 29, 2009 9:00 am Post subject: quote edit **eleven**, When you run suexg-cb with a file parameter: suexg-cb 0 1000 file are you sure it tries each grid in "file" only 1 time to generate 0 or 1 minimal instead of generating 1 minimal per grid (as in the non-cb version)?
Back to top	profile pm www
gsf	Posted: Tue Sep 29, 2009 12:33 pm Post subject: quote **denis_berthier wrote:**

Joined: 22 Sep 2005
Posts: 3850
Location: NJ USA

> Edit: You were speaking of the whole file with 5 billion grids! I was speaking only of my little piece of it. If this is confirmed, then we don't have to care much for automorphisms in statistics.

bingo

Back to top profile pm www

eleven Posted: Tue Sep 29, 2009 4:39 pm Post subject: quote

Joined: 10 Feb 2008
Posts: 500

> **denis_berthier wrote:**
>
> > **eleven**,
> >
> > When you run suexg-cb with a file parameter:
> > suexg-cb 0 1000 file
> > are you sure it tries each grid in "file" only 1 time to generate 0 or 1 minimal instead of generating 1 minimal per grid (as in the non-cb version)?

Oops, no and no.
In the original version with a file given the 2nd parameter means, how often a grid should be used. But this is not working any more in the cb version. For this you have to replace our two "goto m0" by "goto m0s". Then start it with "suexg-cb 0 1 file".

Since yesterday my disc with the linux OS crashed, i am very busy with setting up a new one. So in the moment i cant rewrite it now to work both with a file and grid generation.
Also note, that i had started to add a 4-unavoidable test to the solver. Its not really tested, but hopefully the file version becomes 2 times faster then.

Back to top profile pm

Red Ed Posted: Tue Sep 29, 2009 6:18 pm Post subject: quote

Joined: 06 Jun 2005
Posts: 743

> **gsf wrote:**
>
> > **denis_berthier wrote:**
> >
> > > Edit: You were speaking of the whole file with 5 billion grids! I was speaking only of my little piece of it. If this is confirmed, then we don't have to care much for automorphisms in statistics.
> >
> > bingo

For confirmation of (gsf) "only 560,151 grids have non-trivial automorphisms", see for example kjellfp's post:

Code:

```
N   | Classes [G] where #K(G)=N
    +-------------+------------
    | not trans.- | transpose-
    | invariant   | invariant
----+-------------+------------
  1 | 10944340774 | 23201
  2 |     1050496 |   637
  3 |       14672 |    36
  4 |        4378 |    29
  6 |        2442 |     6
  9 |          84 |     1
```

12	172	0
18	168	4
27	4	1
36	22	2
54	20	1
108	4	0
162	2	0
324	0	1

btw - unrelated return to confidence intervals around the mean number of clues - I tried bootstrap resampling and unsurprisingly it gave *slightly* tighter confidence intervals for the mean (by a factor of 1.84/1.92 IIRC). I'd be happier sampling from a well-fitting distribution, but I doubt I'll have time to do better than the gamma distribution mentioned earlier. Still, CIs by any method are a step forward.

Back to top profile pm

denis_berthier

Posted: Wed Sep 30, 2009 6:12 am Post subject: quote edit

Joined: 19 Jun 2007
Posts: 861
Location: Paris, France

> **eleven wrote:**
> i had started to add a 4-unavoidable test to the solver. Its not really tested, but hopefully the file version becomes 2 times faster then.

Very good!
After some time of using the first accelerated version (46 clues deleted without testing), I can say it is at least 6 (and not 5 as I had said after a quick test) times faster than the non-accelerated one. This is a global ratio, including grid generation. I have no separate figures for the deletion part.

Combining these two improvements with the stream of grids provided by gsf, if he finds a way of hosting his files, should make controlled-bias generation much more appealing.

Back to top profile pm www

denis_berthier

Posted: Wed Sep 30, 2009 6:17 am Post subject: quote edit

Joined: 19 Jun 2007
Posts: 861
Location: Paris, France

> **Red Ed wrote:**
> return to confidence intervals around the mean number of clues - I tried bootstrap resampling and unsurprisingly it gave *slightly* tighter confidence intervals for the mean (by a factor of 1.84/1.92 IIRC).

On my side, I'm still generating puzzles in the hope of getting a better estimate for the 31s. I now have only 2.

Back to top profile pm www

Red Ed

Posted: Wed Sep 30, 2009 6:39 am Post subject: quote

Joined: 06 Jun 2005
Posts: 743

Good news about the second 31. The more the merrier.

Play devil's advocate (*moi?*), have you considered when gsf's very nearly unbiased source is plugged-in to suexg that suexg's MWC() random number generator might become the main source of bias? Might be worth testing or replacing it before

embarking on sweep over the whole set of 5.4bn grids.

Back to top profile pm

denis_berthier

Posted: Wed Sep 30, 2009 7:15 am Post subject: quote edit

Joined: 19 Jun 2007
Posts: 861
Location: Paris, France

Red Ed wrote:

> have you considered when gsf's very nearly unbiased source is plugged-in to suexg that suexg's MWC() random number generator might become the main source of bias? Might be worth testing or replacing it before embarking on sweep over the whole set of 5.4bn grids.

Yes, I've been thinking of this and Mike has also evoked the problem previously. But:
- MWC's cycle is known to be above thousands of billions.
- the probability that MWC's cycle would introduce a cycle in the puzzles generated is very low because a different number of calls to MWC is used every time.

Nevertheless, just to make sure, I systematically test my sets of puzzles against repetitions and auto-correlation.
The only test I don't do (because it'd be too long) is against isomorphisms. But I think the probability of getting 2 isomorphic puzzles in a sample of even 1,000,000 is very low.

The statistical results obtained with the controlled-bias generator call for numerous tests:
- source of complete grids (you and Mike have already done some test with your own sources; gsf's would be another one; I'd like **Allan** to enter the game because he has a very different generator of complete grids)
- RNG (I think Mike uses a different RNG, but I don't remember which)
- not speaking of the various ways of accelerating the algorithm.

Back to top profile pm www

m_b_metcalf

Posted: Wed Sep 30, 2009 7:31 am Post subject: quote

Joined: 15 May 2006
Posts: 2387
Location: Berlin

denis_berthier wrote:

> [- RNG (I think Mike uses a different RNG, but I don't remember which)

RANDOM_NUMBER
Intrinsic Subroutine: Returns one pseudorandom number or an array of such numbers.

Syntax

CALL RANDOM_NUMBER (harvest)

harvest
(Output) Must be of type real. It can be a scalar or an array variable. It is set to contain pseudorandom numbers from the uniform distribution within the range 0 <= x < 1.

The seed for the pseudorandom number generator used by RANDOM_NUMBER can be set or queried with RANDOM_SEED. If RANDOM_SEED is not used, the processor sets the seed for RANDOM_NUMBER to a processor-dependent value.

The RANDOM_NUMBER generator uses two separate congruential generators together to produce a period of approximately 10**18, and produces real pseudorandom results with a uniform distribution in (0,1). It accepts two integer seeds, the first of which is reduced to the range [1, 2147483562]. The second seed is reduced to the range [1, 2147483398]. This means that the generator effectively uses two 31-bit seeds.

For more information on the algorithm, see the following:

Communications of the ACM vol 31 num 6 June 1988, titled: Efficient and Portable Combined Random Number Generators by Pierre L'ecuyer.
Springer-Verlag New York, N. Y. 2nd ed. 1987, titled: A Guide to Simulation by Bratley, P., Fox, B. L., and Schrage, L. E.
See Also

RANDOM_SEED, RANDOM, SEED, DRAND and DRANDM, IRAND and IRANDM, RAN, RAND and RANDOM

Examples

Consider the following:

```
REAL Y, Z (5, 5)
! Initialize Y with a pseudorandom number
CALL RANDOM_NUMBER (HARVEST = Y)
CALL RANDOM_NUMBER (Z)
```

Y and Z contain uniformly distributed random numbers.

The following shows another example:

```
REAL x, array1 (5, 5)
CALL RANDOM_SEED()
CALL RANDOM_NUMBER(x)
CALL RANDOM_NUMBER(array1)
```

Consider also the following:

```
program testrand
intrinsic random_seed, random_number
integer size, seed(2), gseed(2), hiseed(2), zseed(2)
real harvest(10)
data seed /123456789, 987654321/
data hiseed /-1, -1/
data zseed /0, 0/
```

```
call random_seed(SIZE=size)
print *,"size ",size
call random_seed(PUT=hiseed(1:size))
call random_seed(GET=gseed(1:size))
print *,"hiseed gseed", hiseed, gseed
call random_seed(PUT=zseed(1:size))
call random_seed(GET=gseed(1:size))
print *,"zseed gseed ", zseed, gseed
call random_seed(PUT=seed(1:size))
call random_seed(GET=gseed(1:size))
call random_number(HARVEST=harvest)
print *, "seed gseed ", seed, gseed
print *, "harvest"
print *, harvest
call random_seed(GET=gseed(1:size))
print *,"gseed after harvest ", gseed
end program testrand
```

Back to top profile pm

denis_berthier

Posted: Wed Sep 30, 2009 7:42 am Post subject: quote edit

Joined: 19 Jun 2007
Posts: 861
Location: Paris, France

Mike,
thanks for this information.
Is there a C version of this RNG available? (If I remember well, yours is in Fortran).
Or is there any possibility of using directly your Fortran generator (I remember you also programmed a controlled-bias version)?

Back to top profile pm www

m_b_metcalf

Posted: Wed Sep 30, 2009 8:01 am Post subject: quote

Joined: 15 May 2006
Posts: 2387
Location: Berlin

> **denis_berthier wrote:**
>
> Is there a C version of this RNG available? (If I remember well, yours is in Fortran).

Certainly, but I don't know where. A Web search?

> **Quote:**
>
> Or is there any possibility of using directly your Fortran generator (I remember you also programmed a controlled-bias version)?

The RNG is an intrinsic part of the Fortran language and so the implementation is part of the compiler and proprietry, in this case by Intel.

Sorry,

Mike

Back to top profile pm

eleven

Posted: Wed Sep 30, 2009 8:58 am Post subject: quote

Joined: 10 Feb 2008
Posts: 500

I saw, that the C code for the Mersenne Twister RNG by Makoto Matsumoto and Takuji Nishimura is available (mt19937ar.c). it would not be much work to use this (e.g. seed it with 4 numbers of MWC from the command line seed).

Back to top profile pm

Allan Barker

Posted: Wed Sep 30, 2009 1:50 pm Post subject: quote

Joined: 21 Feb 2008
Posts: 342
Location: Bangkok

> **denis_berthier wrote:**
>
> The statistical results obtained with the controlled-bias generator call for numerous tests:
> - source of complete grids (you and Mike have already done some test with your own sources; gsf's would be another one; I'd like Allan to enter the game because he has a very different generator of complete grids)

If you would like, I could generate 1,000,000 grids and put them on my website, or I could paste the code after a bit of clean up. It might also be worthwhile to see if they are any good, maybe **Red Ed** could run his bias tester on them?

The grid generator works like this.

1. Take 81 digits, 9 of each digit from 1 to 9
2. Place the digits in random locations in a grid.
3. Run the Monte Carlo hopper until the grid is valid
___a. Select two different random locations
___b. Score each location by counting duplicate digits in its row, column, and box (a measure of badness)
___c. Swap the two locations
___d. Rescore each location as before.
___e. If the difference is better than criteria keep swap else unswap

Not so fast but generates about 300 grids/sec. For the current purpose I think that would be OK.

Back to top profile pm www

eleven

Posted: Wed Sep 30, 2009 3:20 pm Post subject: quote

Joined: 10 Feb 2008
Posts: 500

This is, what i have now.
First tests said, that reading grids from a file made it about 2.7 times faster. With the pre-check, if a 4-cell-unavoidable set has no given (did about 52% of the solves) its about 4.9 times faster than the old version including grid generation.
But of course there is no warranty, that the program/result is correct.

```
// Program to generate puzzles with controlled bias, see
// http://www.sudoku.com/boards/viewtopic.php?t=14615
// Modified version of the top down generator by dukuso (sterten@aol.com),
// which is public domain
// If no grids file is given, this version prefills the 9 diagonal cells according to Red Ed
// Integrated Mersenne Twister RNG by Makoto Matsumoto and Takuji Nishimura
// 4-unavoidable test added to speed up the solve() function


#include <stdio.h>
```

```
#include <stdlib.h>
#define MWC
((zr=36969*(zr&65535)+(zr>>16))^(wr=18000*(wr&65535)+(wr>>16)))
unsigned zr=362436069, wr=521288629;
int Rows[325],Cols[730],Row[325][10],Col[730][5],Ur[730],Uc[325],V[325],W[325];
int P[88],A[88],A0[88],C[88],I[88],Two[888];
int b,w,f,s1,m0,c1,c2,r1,l,i1,m1,m2,a,p,i,j,k,r,c,d,n=729,m=324,x,y,s;
int mi1,mi2,q7,part,nt,nodes,seed,solutions,min,samples,sam1,sam2,clues;
char L[11]=".123456789";
FILE *file;
int solve();
double cnt = 0.0;
int nClues;


/* *** Mersenne Twister RNG by Makoto Matsumoto and Takuji Nishimura *** */
#define MATRIX_A 0x9908b0dfUL /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

static unsigned long mt[624]; /* the array for the state vector */
static int mti=624+1; /* mti==N+1 means mt[N] is not initialized */

/* initializes mt[624] with a seed */
void init_genrand(unsigned long s)
{
mt[0]= s & 0xffffffffUL;
for (mti=1; mti<624; mti++) {
mt[mti] =
(1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
/* 2002/01/09 modified by Makoto Matsumoto */
mt[mti] &= 0xffffffffUL;
/* for >32 bit machines */
}
}

/* initialize by an array with array-length */
void init_by_array(unsigned long init_key[], int key_length)
{
int i, j, k;
init_genrand(19650218UL);
i=1; j=0;
k = (624>key_length ? 624 : key_length);
for (; k; k--) {
mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL))
+ init_key[j] + j; /* non linear */
mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
i++; j++;
if (i>=624) { mt[0] = mt[624-1]; i=1; }
if (j>=key_length) j=0;
}
for (k=624-1; k; k--) {
mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL))
- i; /* non linear */
mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
i++;
if (i>=624) { mt[0] = mt[624-1]; i=1; }
}

mt[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0,0xffffffff]-interval */
```

```
unsigned long genrand_int32(void)
{
unsigned long y;
static unsigned long mag01[2]={0x0UL, MATRIX_A};
/* mag01[x] = x * MATRIX_A for x=0,1 */

if (mti >= 624) { /* generate 624 words at one time */
int kk;

if (mti == 624+1) /* if init_genrand() has not been called, */
init_genrand(5489UL); /* a default initial seed is used */

for (kk=0;kk<624-397;kk++) {
y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
mt[kk] = mt[kk+397] ^ (y >> 1) ^ mag01[y & 0x1UL];
}
for (;kk<624-1;kk++) {
y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
mt[kk] = mt[kk+(397-624)] ^ (y >> 1) ^ mag01[y & 0x1UL];
}
y = (mt[624-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
mt[624-1] = mt[397-1] ^ (y >> 1) ^ mag01[y & 0x1UL];

mti = 0;
}

y = mt[mti++];

/* Tempering */
y ^= (y >> 11);
y ^= (y << 7) & 0x9d2c5680UL;
y ^= (y << 15) & 0xefc60000UL;
y ^= (y >> 18);

return y;
}

unsigned randInt( unsigned n )
{
// Find which bits are used in n
// Optimized by Magnus Jonsson (magnus@smartelectronix.com)
unsigned used = n;
used |= used >> 1;
used |= used >> 2;
used |= used >> 4;
used |= used >> 8;
used |= used >> 16;

// Draw numbers until one is found in [0,n]
unsigned i;
do
i = genrand_int32() & used; // toss unused bits to shorten search
while( i > n );
return i;
}

/* *** calc and test for 4-cell-unavoidables *** */
int nr4unavoid;
int a4unavoid[64][4];
void add4unavoid(int i, int j, int k, int l)
{
a4unavoid[nr4unavoid][0]=i; a4unavoid[nr4unavoid][1]=j;
a4unavoid[nr4unavoid][2]=k; a4unavoid[nr4unavoid][3]=l;
```

```
nr4unavoid++;
}
void find4unavoid(int i, int j)
{
int k,l;
k=1+9*((i-1)%9)+(i-1)/9;l=1+9*((j-1)%9)+(j-1)/9;
if (A[i]==A[j+9]&&A[j]==A[i+9])add4unavoid(i,j,i+9,j+9);
else if (A[i]==A[j+18]&&A[j]==A[i+18])add4unavoid(i,j,i+18,j+18);
else if (A[i+9]==A[j+18]&&A[j+9]==A[i+18])add4unavoid(i+9,j+9,i+18,j+18);
else if (A[k]==A[l+1]&&A[l]==A[k+1])add4unavoid(k,l,k+1,l+1);
else if (A[k]==A[l+2]&&A[l]==A[k+2])add4unavoid(k,l,k+2,l+2);
else if (A[k+1]==A[l+2]&&A[l+1]==A[k+2])add4unavoid(k+1,l+1,k+2,l+2);
}
int calc4unavoid()
{
int i,j,i1,k,l;
nr4unavoid = 0;
for (i1=1;i1<81;i1+=27){
for (i=i1;i<i1+3;i++){
for (j=i1+3;j<i1+9;j++){find4unavoid(i,j);}}
for (i=i1+3;i<i1+6;i++){
for (j=i1+6;j<i1+9;j++){find4unavoid(i,j);}}}
return nr4unavoid;
}
int empty4unavoid()
{
int i;
for (i=0;i<nr4unavoid;i++)
if (!A[a4unavoid[i][0]] && !A[a4unavoid[i][1]] && !A[a4unavoid[i][2]] &&
!A[a4unavoid[i][3]])
return 1;
return 0;
}

/* *** in grid generation mode prefill the 9 diagonal cells *** */
void prefill()
{
int i,k,o[10];
for(i=1;i<=9;i++){k=randInt(i-1);k++;o[i]=o[k];o[k]=i;}
for (i=0;i<3;i++) A[1+10*i]=o[i+1];
for(i=1;i<=9;i++){k=randInt(i-1);k++;o[i]=o[k];o[k]=i;}
for (i=0;i<3;i++) A[31+10*i]=o[i+1];
for(i=1;i<=9;i++){k=randInt(i-1);k++;o[i]=o[k];o[k]=i;}
for (i=0;i<3;i++) A[61+10*i]=o[i+1];
}

/* *** main *** */
int main(int argc,char*argv[]){
if(argc<3)
{ printf("\nusage:\n");
printf("<program name> random-seed max-puzzles \n");
printf("<program name> random-seed max-tries-per-grid file-with-grids\n\n");
return(1);}

sscanf(argv[1],"%i",&seed);zr^=seed;wr+=seed;
sscanf(argv[2],"%i",&samples);
if(argc>3)if((file=fopen(argv[3],"rb"))==NULL)
{printf("\ncan't find file %s\n",argv[3]);return(1);}

for(i=0;i<888;i++){j=1;while(j<=i)j+=j;Two[i]=j-1;}
for(i=1;i<=81;i++)A0[i]=0;

r=0;for(x=1;x<=9;x++)for(y=1;y<=9;y++)for(s=1;s<=9;s++){
```

```
r++;Cols[r]=4;k=3;
Col[r][1]=x*9-9+y;
Col[r][2]=(k*((x-1)/k)+(y-1)/3)*9+s+81*1;
Col[r][3]=x*9-9+s+81*2;
Col[r][4]=y*9-9+s+81*3;}
for(c=1;c<=m;c++)Rows[c]=0;
for(r=1;r<=n;r++)for(c=1;c<=Cols[r];c++){
a=Col[r][c];Rows[a]++;Row[a][Rows[a]]=r;}

// init RNG
unsigned long init[4]={MWC, MWC, MWC, MWC}, length=4;
init_by_array(init, length);

m6:if(argc>3)
for(i=1;i<=81;i++){
m6a:A0[i]=fgetc(file)-48;if(feof(file))return(8);
if(A0[i]==-2)A0[i]=0;if(A0[i]<0 || A0[i]>9)goto m6a;}

sam1=0; sam2=0;
m0s:if(argc>3){sam1++;if(sam1>samples) goto m6;}else
if(sam2>=samples)return(0);

m0:cnt+=1.0;for(i=1;i<=81;i++)A[i]=A0[i];part=0;if(argc<4){prefill();solve();}
nClues=81;

// save all 4-cell anavoidables in the grid
calc4unavoid();
part++;

for(i=1;i<=81;i++){x=randInt(i-1);x++;P[i]=P[x];P[x]=i;}
for(i1=1;i1<=81;i1++){s1=A[P[i1]];if(s1){A[P[i1]]=0;
if(--nClues==34){if(solve()>1)goto
m0s;}if(nClues<34&&solve()>1){A[P[i1]]=s1;break;}}}

i=++i1;for(i1=i;i1<=81;i1++){s1=A[P[i1]];if(s1){A[P[i1]]=0;if(solve()<2)goto m0s;
A[P[i1]]=s1;}}


for(i=1;i<=81;i++)printf("%c",L[A[i]]);printf(" %g\n", cnt);
cnt=0; // dont sum it up for all puzzles
if(argc<4)sam2++;
fflush(stdout);
goto m0s;}

/* *** solver code *** */
int solve(){
// test for 4-cell unavoidables without givens
if (part && empty4unavoid())
return 2;

for(i=0;i<=n;i++)Ur[i]=0;for(i=0;i<=m;i++)Uc[i]=0;
clues=0;for(i=1;i<=81;i++)
if(A[i]){clues++;r=i*9-9+A[i];
for(j=1;j<=Cols[r];j++){d=Col[r][j];if(Uc[d])return -1;Uc[d]++;
for(k=1;k<=Rows[d];k++){Ur[Row[d][k]]++;}}}
for(c=1;c<=m;c++){V[c]=0;for(r=1;r<=Rows[c];r++)if(Ur[Row[c][r]]==0)V[c]++;}
i=clues;m0=0;m1=0;solutions=0;if(i==81)return 1;
m2:i++;I[i]=0;min=n+1;if(i>81 || m0)goto m4;
if(m1){C[i]=m1;goto m3;}
w=0;for(c=1;c<=m;c++)if(!Uc[c]) { if(V[c]<2){C[i]=c;goto m3;}
if(V[c]<=min){w++;W[w]=c;};
if(V[c]<min){w=1;W[w]=c;min=V[c];} }
mr:c2=genrand_int32()&Two[w];if(c2>=w)goto mr;C[i]=W[c2+1];
```

```
m3:c=C[i];I[i]++;if(I[i]>Rows[c])goto m4;
r=Row[c][I[i]];if(Ur[r])goto m3;m0=0;m1=0;
if(part==0){j=9;k=81;x=(r-1)/k+1;y=((r-1)%k)/j+1;s=(r-1)%j+1;A[x*9-
9+y]=s;P[x*9-9+y]=i;}
for(j=1;j<=Cols[r];j++){c1=Col[r][j];Uc[c1]++;}
for(j=1;j<=Cols[r];j++){c1=Col[r][j];
for(k=1;k<=Rows[c1];k++){r1=Row[c1][k];Ur[r1]++;
if(Ur[r1]==1)for(l=1;l<=Cols[r1];l++){c2=Col[r1][l];V[c2]--;
if(Uc[c2]+V[c2]<1)m0=c2;if(Uc[c2]==0 && V[c2]<2)m1=c2;}}}
if(i==81){solutions++;if(solutions>1)return 2;if(part==0)return 1;}
goto m2;
m4:i--;c=C[i];r=Row[c][I[i]];if(i==clues)goto m9;
for(j=1;j<=Cols[r];j++){c1=Col[r][j];Uc[c1]--;
for(k=1;k<=Rows[c1];k++){r1=Row[c1][k];Ur[r1]--;
if(Ur[r1]==0)for(l=1;l<=Cols[r1];l++){c2=Col[r1][l];V[c2]++;}}}
if(i>clues)goto m3;
m9:return solutions;}
```

Back to top profile pm

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 ... , 24, 25, 26 Next

Page 25 of 26

Stop watching this topic

Jump to: General/puzzle Go