May 19, 2018 at 02:30

1. Intro. Here I try to generate SAT instances for partial van der Waerden configurations: There are to be no t equally spaced ones, in $x_1 \ldots x_n$. Furthermore I try to make $x_1 + \cdots + x_n = r$.

One key idea is to use the Sinz-inspired clauses with $(n-r)r$ auxiliary variables

$$s_j^k = [x_1 + \cdots + x_{j+k-1} \geq k], \qquad \text{for } 1 \leq j \leq n-r \text{ and } 1 \leq k \leq r.$$

[See *Lecture Notes in Computer Science* **3709** (2005), 827–831.] The clauses are

$$\begin{aligned}
\bar{s}_j^k \vee s_{j+1}^k&, \text{ for } 1 \leq j < n-r \text{ and } 1 \leq k \leq r;\\
s_j^k \vee \bar{s}_j^{k+1}&, \text{ for } 1 \leq j \leq n-r \text{ and } 1 \leq k < r;\\
\bar{x}_{j+k} \vee \bar{s}_j^k \vee s_j^{k+1}&, \text{ for } 1 \leq j \leq n-r \text{ and } 0 \leq k \leq r;\\
x_{j+k} \vee s_j^k \vee \bar{s}_{j+1}^k&, \text{ for } 0 \leq j \leq n-r \text{ and } 1 \leq k \leq r.
\end{aligned}$$

(And we simplify them at the boundaries by using $s_0^k = s_j^{r+1} = 0$, $s_j^0 = s_{n-r+1}^k = 1$.)

This program uses the code 'jSk' to denote s_j^k in its output.

I assume the existence of a file `free`t`.dat` that contains the smallest values n_1, ..., n_{r-1} of n for which this problem is satisfiable for smaller values of r. For example, `free3.dat` begins with the numbers 1, 2, 4, 5, 9, 11, 13, 14; and I might be running this program with $t = 3$, $n = 18$, $r = 9$ to see if the number 18 should come next in that file. [Answer: The clauses to be generated are unsatisfiable; therefore the next number in the file must be at least 19. In fact, the next number $F_3(9)$ turns out to be 20.]

Continuing that example, we know that $x_{a+1}+x_{a+2}+x_{a+3} \leq 2$ for $0 \leq a \leq 15$; similarly $x_{a+1}+\cdots+x_{a+8} \leq 4$, $x_{a+1} + \cdots + x_{a+10} \leq 5$, $x_{a+1} + \cdots + x_{a+12} \leq 6$, and $x_{a+1} + \cdots + x_{a+17} \leq 8$, according to the previously tabulated numbers on file. (Two other inequalities, $x_{a+1} + \cdots + x_{a+4} \leq 3$ and $x_{a+1} + \cdots + x_{a+13} \leq 7$, also belong to this pattern; but they are trivial consequences of their predecessors.)

In general if $x_{a+1}+\cdots+x_{a+p} \leq q$ for $0 \leq a \leq n-p$, we can convert that information into useful constraints on the auxiliary variables s_j^k, because $x_1 + \cdots + x_{a+p} \geq k$ implies that $x_1 + \cdots + x_a \geq k - q$. If we set $b = a - k + q + 1$, we can rewrite this statement as "s_{b+p-q}^k implies s_b^{k-q}"; that is, the clauses $\bar{s}_{b+p-q}^k \vee s_b^{k-q}$ must be valid, for $0 \leq b \leq n-r+1-p+q$ and $q < k \leq r$.

For example, when $p = 17$ and $q = 8$, there are just two special clauses, $\bar{s}_9^9 \vee s_0^1$ and $\bar{s}_{10}^9 \vee s_1^1$, which simplify to $\bar{s}_9^9$ and s_1^1. Equivalently, $x_{18} = 1$ and $x_1 = 1$. (Similar deductions will always occur, when we're trying to establish extreme values; we'll necessarily have $x_1 = x_n = 1$, since the case $n-1$ admits at most $r-1$ 1s.)

This program first generates the clauses in $\neg x_i$ that enforce the no-k-equally-spaced-ones constraints. Then it generates the clauses above, and one more to reduce symmetry.

In the example with $n = 18$ and $r = 9$, the Sinz-plus-subinterval clauses themselves (without the arithmetic progression clauses) already force most of the variables s_j^k: By unit propagations they are

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
 & & & 1 & 1 & 1 & 1 & 1 & 1\\
0 & & & 1 & 1 & 1 & 1 & 1 & 1\\
0 & & & & 1 & 1 & 1 & 1 & 1\\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1\\
0 & 0 & 0 & 0 & 0 & & & & 1\\
0 & 0 & 0 & 0 & 0 & 0 & & & 1\\
0 & 0 & 0 & 0 & 0 & 0 & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}.$$

```
#define maxr 100
#include <stdio.h>
#include <stdlib.h>
  FILE *infile;
```

```
int table[maxr + 1];
int t, n, r;    /* command-line parameters */
char buf[16];

main(int argc, char *argv[])
{
  register int b, d, i, j, k, p, q;
  ⟨Process the command line 2⟩;
  ⟨Input the file freet 3⟩;
  printf("~ sat-arithprog %d %d %d\n", t, n, r);
  ⟨Output the negative clauses 4⟩;
  ⟨Output the standard Sinz clauses 5⟩;
  ⟨Output the special implications 6⟩;
  ⟨Output the final symmetry-breaking clause 7⟩;
}
```

2. ⟨Process the command line 2⟩ ≡

```
if (argc ≠ 4 ∨ sscanf(argv[1], "%d", &t) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1 ∨ sscanf(argv[3], "%d", &r) ≠ 1)
  {
  fprintf(stderr, "Usage: %s t n r\n", argv[0]);
  exit(−1);
}
if (r > maxr) {
  fprintf(stderr, "Sorry, r (%d) should not exceed %d!\n", r, maxr);
  exit(−2);
}
if (n ≥ 256) {
  fprintf(stderr, "Sorry, n (%d) must not exceed 255!\n", n);
  exit(−3);
}
```

This code is used in section 1.

3. ⟨Input the file freet 3⟩ ≡

```
sprintf(buf, "free%d.dat", t);
infile = fopen(buf, "r");
if (¬infile) {
  fprintf(stderr, "I can't open file `%s' for reading!\n", buf);
  exit(−5);
}
for (j = 1; j < r; j++) {
  if (fscanf(infile, "%d", &table[j]) ≠ 1) {
    fprintf(stderr, "I couldn't read item %d in file `%s'!\n", j, buf);
    exit(−6);
  }
}
table[r] = n;
```

This code is used in section 1.

4. ⟨ Output the negative clauses 4 ⟩ ≡

```
for (d = 1; 1 + (t − 1) ∗ d ≤ n; d++) {
  for (i = 1; i + (t − 1) ∗ d ≤ n; i++) {
    for (j = 0; j < t; j++) printf("␣~x%d", i + j ∗ d);
    printf("\n");
  }
}
```

This code is used in section 1.

5. ⟨ Output the standard Sinz clauses 5 ⟩ ≡

```
for (j = 1; j < n − r; j++)
  for (k = 1; k ≤ r; k++) printf("~%dS%d␣%dS%d\n", j, k, j + 1, k);
for (j = 1; j ≤ n − r; j++)
  for (k = 1; k < r; k++) printf("%dS%d␣~%dS%d\n", j, k, j, k + 1);
for (j = 1; j ≤ n − r; j++)
  for (k = 0; k ≤ r; k++) {
    printf("~x%d", j + k);
    if (k > 0) printf("␣~%dS%d", j, k);
    if (k < r) printf("␣%dS%d", j, k + 1);
    printf("\n");
  }
for (j = 0; j ≤ n − r; j++)
  for (k = 1; k ≤ r; k++) {
    printf("x%d", j + k);
    if (j > 0) printf("␣%dS%d", j, k);
    if (j < n − r) printf("␣~%dS%d", j + 1, k);
    printf("\n");
  }
```

This code is used in section 1.

6. ⟨ Output the special implications 6 ⟩ ≡

```
for (q = 2; q < r; q++)
  if (table[q + 1] > table[q] + 1) {
    p = table[q + 1] − 1;
    for (b = 0; b ≤ n − r + 1 − p + q; b++)
      for (k = q + 1; k ≤ r; k++) {
        if (b + p − q ≤ n − r) printf("~%dS%d", b + p − q, k);
        if (b > 0) printf("␣%dS%d", b, k − q);
        printf("\n");
      }
  }
```

This code is used in section 1.

7. The left-to-right reflection of any solution is also a solution. Therefore we can conclude that any solution with $s_{\lceil (n-r)/2\rceil}^{\lceil r/2\rceil}=1$ implies a solution with $s_{\lceil (n-r)/2\rceil}^{\lceil r/2\rceil}=0$, if r is odd. On the other hand if n and r are both even, we can assume that $x_1+\cdots+x_{n/2}\geq r/2$.

⟨ Output the final symmetry-breaking clause 7 ⟩ ≡

```
if (r & 1) {
  j = (n − r + 1) ≫ 1, k = (r + 1) ≫ 1;
  printf("~%dS%d\n", j, k);
} else if (¬(n & 1)) {
  j = (n − r) ≫ 1, k = r ≫ 1;
  printf("%dS%d\n", j + 1, k);
}
```

This code is used in section 1.

8. Index.

SAT-ARITHPROG