1. Intro. This program tries to generate a "hard" open shop scheduling problem with n jobs and n machines, using the method suggested by Guéret and Prins in *Annals of Operations Research* **92** (1999), 165–183: We start with work times w_{ij} that are as near equal as possible, having constant row and column sums s. Then we choose random rows $i \neq i'$ and random columns $j \neq j'$, and transfer δ units of weight by setting

$$w_{ij} \leftarrow w_{ij} - \delta, \quad w_{i'j} \leftarrow wi'j + \delta, \quad w_{ij'} \leftarrow wij' + \delta, \quad w_{i'j'} \leftarrow wi'j' - \delta,$$

where $\delta \geq w_{ij}$ and $\delta \geq w_{i'j'}$; this operation clearly preserves the row and column sums. The value of δ is randomly distributed between $p \min\{w_{ij}, w_{i'j'}\}$ and $\min\{w_{ij}, w_{i'j'}\}$, where p is a parameter. The final weights are obtained after making r such transfers.

Parameters n, s, p, and r are given on the command line, together with a random seed value to specify the source of random numbers. (Guéret and Prins suggested taking $p = .95$ when $n \geq 6$, and $r = n^3$.)

The output consists of n lines of n numbers each, suitable for input to SAT-OSS.

```
#define maxn '~' - '0'    /* jobs/machines are single characters, '0' ≤ c < '~' */
#include <stdio.h>
#include <stdlib.h>
#include "gb_flip.h"    /* the random number generator */
  int n, s, r, seed;    /* integer command-line parameters */
  float p;    /* the floating point command-line parameter */
  int w[maxn][maxn];    /* the work times */
  main(int argc, char *argv[])
  {
    register int i, j, ii, jj, del, max_take, rep;
    ⟨ Process the command line 2 ⟩;
    ⟨ Create the initial weights 3 ⟩;
    for (rep = 0; rep < r; rep++) ⟨ Make a random weight transfer 4 ⟩;
    ⟨ Output the final weights 5 ⟩;
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
  if (argc ≠ 6 ∨ sscanf(argv[1], "%d", &n) ≠ 1 ∨ sscanf(argv[2], "%d", &s) ≠ 1 ∨ sscanf(argv[3], "%g",
        &p) ≠ 1 ∨ sscanf(argv[4], "%d", &r) ≠ 1 ∨ sscanf(argv[5], "%d", &seed) ≠ 1) {
    fprintf(stderr, "Usage: %s n scale prob reps seed\n", argv[0]);
    exit(−1);
  }
  if (p < 0 ∨ p ≥ 1.0) {
    fprintf(stderr, "The probability must be between 0.0 and 1.0, not %.2g!\n", p);
    exit(−2);
  }
  gb_init_rand(seed);
  printf("~ oss-data %d %d %g %d %d\n", n, s, p, r, seed);
```

This code is used in section 1.

3. ⟨ Create the initial weights 3 ⟩ ≡

```
  del = s/n;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) w[i][j] = del;
  del = s − n * del;
  for (i = 0; i < n; i++)
    for (j = 0; j < del; j++) w[i][(i + j) % n]++;
```

This code is used in section 1.

4. ⟨ Make a random weight transfer 4 ⟩ ≡

```
{
  while (1) {
    i = gb_unif_rand(n);
    ii = gb_unif_rand(n);
    if (i ≠ ii) break;
  }
  while (1) {
    j = gb_unif_rand(n);
    jj = gb_unif_rand(n);
    if (j ≠ jj) break;
  }
  del = (w[i][j] ≤ w[ii][jj] ? w[i][j] : w[ii][jj]);
  max_take = (1 − p) ∗ (float) del;
  if (max_take) del −= gb_unif_rand(max_take);
  w[i][j] −= del;
  w[ii][j] += del;
  w[i][jj] += del;
  w[ii][jj] −= del;
}
```

This code is used in section 1.

5. ⟨ Output the final weights 5 ⟩ ≡

```
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) printf("␣%d", w[i][j]);
  printf("\n");
}
```

This code is used in section 1.

6. Index.

OSS-DATA