The Lost Group Chart and Related Problems*

Tom Verhoeff

1 Introduction

Last year, Marga and I bought ourselves a new home. This solved our problem of room shortage but created others ...

In the basement of the house is a fuse box, where the main electricity supply is distributed over ten separately fused groups. Each fuse protects a number of the—more than fifty—power outlets in and around the house. For reasons that need not concern us here, the mapping from outlet to fuse group is rather haphazard. Unfortunately, the previous owner had lost the group chart that lists for every outlet the fuse group to which it belongs. Thus, I encountered the problem of reconstructing the lost group chart.

The fuse box in the basement also has ten switches, one for each group. All outlets in a group are connected to the mains supply through the corresponding switch. A voltage tester can be used to determine whether an outlet is 'alive' or not. By suitably toggling switches and testing outlets it should be possible to reconstruct the group chart. Because I like brain teasers better than manual labor, I imposed the additional constraint that the number of switch toggles and outlet tests be minimized.

2 Specification

The reconstruction problem can be formulated as a programming exercise in the following way (also see Figure 1). M outlets labeled 1 through M (at A) are connected by M wires to N switches labeled 1 through N (at B). Each outlet is connected to exactly one of the switches. Each switch can be connected to zero[1] or more wires.

The reconstruction procedure has to determine how the outlets are connected to the switches using the following two kinds of operations. Each switch can be *toggled* from a conducting state to a non-conducting state and vice versa. Initially all switches are, say, non-conducting.

*Appeared in: *Simplex Sigillum Veri*, Een Liber Amicorum voor prof. dr. F. E. J. Kruseman Aretz. Eindhoven University of Technology, Faculty of Mathematics and Computing Science. December 1995. (pp. 308–313)

[1] This takes into account the possibility of unused fuses.

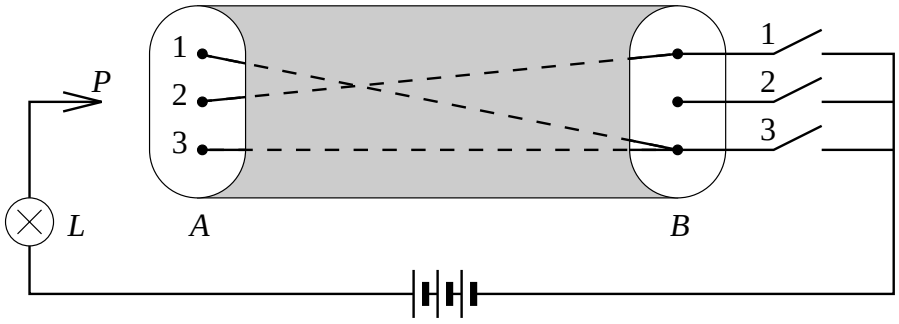

Figure 1: Three outlets connected to three switches ($M = N = 3$)

An outlet can be *tested* with probe P: lamp L will light up if and only if the tested outlet is connected to a conducting switch.

A more formal specification of the programming exercise is given below in a dialect of the programming language Pascal.

const
  M: *integer* = ...; { number of outlets, $1 \leq M$ }
  N: *integer* = ...; { number of switches, $1 \leq N$ }

type
  outlet = 1..M;
  switch = 1..N;
  chart = **array** [*outlet*] **of** *switch*;

const
  f: *chart* = ...; { the group chart to be reconstructed }

var
  C: **set of** *switch*; { the set of conducting switches }

function *test*(i: *outlet*): *boolean*; { return: $f[i] \in C$ }

procedure *toggle*(j: *switch*); { pre: $C = C'$; post: $C = C' \div [j]$ }

procedure *reconstruct*(**var** g: *chart*); { pre: $C = [\,]$; post: $g = f$ }

Function call *test*(i) returns whether the lamp lights up when testing outlet i. Procedure call *toggle*(j) changes the state of switch j (operator $\div$ stands for the symmetric set difference). Procedure *reconstruct* is to be programmed using only *test* to inspect f and *toggle* to modify C, while minimizing the number of calls of *test* and *toggle*.

3 Straightforward Solution

A straightforward design for procedure *reconstruct* determines for each outlet the switch to which that outlet is connected, based on the property

$$C = [j] \wedge \mathit{test}(i) \quad \Rightarrow \quad f[i] = j$$

Here is the code (the heading of *reconstruct* is not be repeated):

```
var i: outlet; j: switch;
begin
for i := 1 to M do { inv: C = [ ] ∧ (∀ a : 1 ≤ a < i : g[a] = f[a]) }
  for j := 1 to N do begin
    toggle(j) ; { C = [j] }
    if test(i) then g[i] := j ;
    toggle(j)
    end { for j }
end; { reconstruct }
```

This procedure does $2MN$ toggles and MN tests. The average-case performance can be improved by breaking off the inner loop (over j) as soon as $g[i]$ has been determined. This would halve the expected number of toggles and tests.

Another improvement is obtained by reordering the two loops and taking the calls to *toggle* outside the inner loop, that is, by determining for each switch all outlets that are connected to that switch:

```
var i: outlet; j: switch;
begin
for j := 1 to N do begin
  { inv: C = [ ] ∧ (∀ i : f[i] < j : g[i] = f[i]) }
  toggle(j) ;
  for i := 1 to M do { inv: C = [j] }
    if test(i) then g[i] := j ;
  toggle(j)
  end { for j }
end; { reconstruct }
```

The procedure now does $2N$ toggles and MN tests. The average-case performance can be improved by not testing outlets that have already been determined. This also removes the need to reset the switches. Furthermore, when $N-1$ switches have been covered the remaining outlets are known to be connected to switch N. The following design incorporates these three improvements. It is based on the property

$$C = [1..j] \wedge f[i] \geq j \wedge \mathit{test}(i) \quad \Rightarrow \quad f[i] = j$$

Below is the—surprisingly compact—code. The number of toggles is now $N-1$, the bare minimum under worst-case conditions. The expected number of tests is $\frac{1}{2}M(N-1)$.

```
var i: outlet; j: switch;
begin
for i := 1 to M do g[i] := N ;
for j := 1 to N−1 do begin
  { inv: C = [1..j−1] ∧ (∀ i :: g[i] = if f[i] < j then f[i] else N) }
  toggle(j) ;
  for i := 1 to M do { inv: C = [1..j] }
    if g[i] = N then { f[i] ≥ j }
      if test(i) then g[i] := j
  end { for j }
end; { reconstruct }
```

4 Sophisticated Solution

The total number of possible charts is N^M. Because each test provides at most one bit of information, the worst-case minimum number of tests to reconstruct the chart is $\lceil M \log_2 N \rceil$. This analysis can be refined as follows.

The knowledge gathered about f during reconstruction can be captured by stating for each outlet to what *set* of switches it is possibly connected. Let me denote the candidate set for outlet i by $F(i)$. Initially, $F(i) = [1..N]$ for all i. If testing outlet i yields *true*, its set of candidate switches is reduced to $F(i) \cap C$. If the test yields *false*, the candidate set reduces to $F(i) - C$. Testing continues until all $F(i)$ are singletons. The best one can hope to accomplish by one test is halving the outlet's candidate set. Therefore, the worst-case minimum number of tests to reduce one candidate set to a singleton is $\lceil \log_2 N \rceil$. This lower bound can be attained by a binary search:

```
procedure BinarySearch(i: outlet); { post: g[i] = f[i] }
var L, R, j, k: switch; p: boolean;
begin
L := 1 ; R := N ; p := false ; { L ≤ f[i] ≤ R, switches [L..R] are p-conducting }
while L ≠ R do begin
  k := (L + R − 1) div 2 ; { L ≤ k ≤ R, #[L..k] ≤ #[k+1..R] }
  for j := L to k do toggle(j) ;
  { switches [L..k] are ¬p-conducting, [k+1..R] are p-conducting }
  if test(i) = p then { f[i] ∈ [k+1..R] } L := k+1
  else { f[i] ∈ [L..k] } begin R := k ; p := not p end
  end { while } ;
g[i] := L
end; { BinarySearch }
```

Note that to reduce the number of *toggle* calls the shorter half of the interval is toggled. Reconstructing the whole chart requires at least $M\lceil \log_2 N \rceil$ tests (under worst-case conditions) and this can indeed be accomplished by combining all M binary searches. For each outlet the *interval* of candidate switches is maintained in an array h:

```
var i: switch;
    h: array [outlet] of record hL, hR: switch end;
    { h[i].hL ≤ f[i] ≤ h[i].hR }
begin
for i := 1 to M do begin h[i].hL := 1 ; h[i].hR := N end ;
CBS(1, N, false, M) ;
for i := 1 to M do g[i] := h[i].hL
end; { reconstruct }
```

Recursive procedure *CBS* halves candidate intervals until they are singletons, minimizing the total number of calls to *toggle* and *test*:

```
procedure CBS(L , R: switch; p: boolean; t: integer);
  { pre: L ≤ R, switches [L..R] are p-conducting and service t outlets }
  var i: outlet; j, k: switch; u: integer;
  begin
  if (L ≠ R) and (t > 0) then begin
    k := (L + R − 1) div 2 ;
    for j := L to k do toggle(j) ;
    { switches [L..k] are ¬p-conducting, [k+1..R] are p-conducting }
    u := 0 ; { switches [L..k] service u outlets in [1..i−1] }
    for i := 1 to M do with h[i] do
      if (hL = L) and (hR = R) then
        if test(i) = p then hL := k+1 ;
        else begin hR := k ; u := u+1 end
    CBS(L , k, not p, u) ; CBS(k+1, R, p, t−u)
    end { if }
  end; { CBS }
```

The following table summarizes the intervals that occur when doing a combined binary search involving ten switches:

phases	switches →										#toggles
↓	1	2	3	4	5	6	7	8	9	10	↓
1	●	●	●	●	●	○	○	○	○	○	5
2	○	○	●	●	●	●	●	○	○	○	4
3	●	○	○	●	●	○	●	●	○	○	4
4				○	●				●	○	2
#toggles →	3	2	2	2	1	2	1	1	1	0	15

For instance, in phase 1, switches [1..5] are made conducting (marked •) and switches [6..10] are kept non-conducting (marked ∘). In total, at most 15 toggles are done. In general the number of toggles is approximately $\frac{1}{2}N \log_2 N$, since in each phase at most half the switches are toggled.

Parameter t and variable u were introduced to suppress superfluous toggles. To understand their effect consider a chart f for ten switches (see table above) where no outlets are connected to switches [1..2]. This will already be detected in phase 2, when invoking $CBS(1, 2, \mathit{false}, 0)$. Without parameter t, switch 1 would still be toggled. For the worst-case situation with ten switches, *CBS* either saves a toggle (when no outlet is connected to switches [1..2]) or a test (when some outlet is connected to switches [1..2]) compared to omitting parameter t.

5 Practical Solution

Let me consider a slightly more general problem, where chart f is a *partial* function from outlets to switches. That is, outlets need not be connected at all (possibly due to broken wires, which is not unrealistic in older houses). The specification is modified as follows:

```
type
  switch′ = 0..N;
  chart = array [outlet] of switch′;
  { for f: chart, f[i] = 0 means that outlet i is not connected }
```

The earlier solutions can easily be adapted. The following solution is of interest because it is efficient and can be carried out with a minimum of administrative overhead. It is based on writing the switch numbers in binary notation. Each bit is used to determine the switch state in the corresponding test phase. Each test yields one bit of the switch number of the tested outlet. Here is the code:

```
var i: outlet; j: switch; k: integer;
begin
for i := 1 to M do g[i] := 0 ;
k := 1 ; while k ≤ N do k := 2 ∗ k ;
while k ≠ 1 do begin
  k := k div 2 ;
  for j := 1 to N do
    if odd(j div k) ≠ (j in C) then toggle(j) ;
  for i := 1 to M do
    if test(i) then g[i] := g[i] + k
  end { while }
end; { reconstruct }
```

Note that $odd(j \,\mathbf{div}\, k)$ yields the bit of weight k in j. The number of tests is $M\lceil \log_2(N+1) \rceil$. The number of toggles is slightly more than the binary search method of the preceding section. For practical reasons I have used the binary notation method at home.

6 Conclusion

The reconstruction problem (with $M = N \leq 90$) appeared as a programming task at the 7th International Olympiad in Informatics[2] which was held at Eindhoven University of Technology in 1995. There were more than two hundred participants from secondary schools of some fifty countries all over the world. To my surprise ten competitors came up with the intended optimized solution (doing a combined binary search and suppressing superfluous toggles).

The problem has some intriguing variants. I already mentioned the extension from total to partial charts f. But what if, for instance, the initial state of the switches is unknown and C may not be inspected? This corresponds to a randomized fuse box where the visible state of the switches does not reveal their conductivity.

What if each wire is also connected to a unique switch on side A, allowing multiple wires to be tested? In that case the minimum number of tests (under worst-case conditions) is conjectured to be $\lceil M \log_2 N \rceil$ instead $M\lceil \log_2 N \rceil$. For instance, for $M = N = 3$ there are 27 possible charts. With switches on side B only, at least $3\lceil \log_2 3 \rceil = 6$ tests are required. With additional switches on side A, it can be done in $\lceil 3 \log_2 3 \rceil = 5$ tests (try it).

What if chart f is known to be a permutation? This situation arises when recycling cables with indistinguishable wires. Can the reconstruction be done in $\lceil \log_2 N! \rceil$ tests? I have the feeling that this variant is as hard as minimum-comparison sorting. Would additional switches on side A help?

When considered carefully, the reconstruction problem produces little programs that read like lovely poems. I would like to thank Frans, whom I have always enjoyed as an excellent educator, for helping me develop a taste for such poetry.

[2] For more information about the IOI see URL http://olympiads.win.tue.nl/ioi/