Educated brute-force to get $\mathbf{h}(4)$

Rogério Reis Nelma Moreira João Pedro Pedroso

Technical Report Series: DCC-04-04 rev.3

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

---

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

Educated brute-force to get $\mathbf{h}(4)$

Rogério Reis Nelma Moreira João Pedro Pedroso
{rvr,nam,jpp}@ncc.up.pt
DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 823, 4150 Porto, Portugal

June 2004

Abstract

In one of his numerous conferences, Frank Harary, talked about one of his many games, that, as usual, had a very difficult problem associated to it. In this case, a family of games for two players in which the selected number of columns in the game has a vital importance. He has proved that for 2 and 3 columns the longest match has 9 and 24 moves respectively, that is to say that $\mathbf{h}(2) = 9$ and $\mathbf{h}(3) = 24$. At the same time it was announced that he knew a solution of length 66 for the problem with 4 columns, but he didn't know if it was the maximum. We present here a program that proves that $\mathbf{h}(4) = 67$. Although it uses but a brute-force approach, its soundness seems good fun to prove.

1 The name of the game

One of the many games that Frank Harary presented in his talks, consists of a small fixed number of empty columns in which each player places, in his turn and beginning with the number 1, the next integer. The column in which the number is placed must be such that it is not possible to write it **as the sum of two other elements of that column**. The player that cannot complete his move complying to these rules, looses.

Here is an example of a game with 3 columns, in which *player A* starts choosing column C_1 and wins because *player B* cannot place number 12 in any column:

$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$
1	**3**	**4**
2	5	6
7	**9**	**8**
10		11

This game cannot proceed. For each number of columns is there a longer game possible? If it exists, say for c columns, we call it the Harary number of order c or $\mathbf{h}(c)$. Putting this in a more formal manner... To simplify let us adopt the following notation

$$\mathbb{N}_n = \{0, \ldots, n\}$$

and

$$\mathbb{N}_n^+ = \{1, \ldots, n\}.$$

Then we can define $\mathbf{h}(c)$ as

$$1 + \mathbf{max}\{k \in \mathbb{N} \mid \exists f : \mathbb{N}_k^+ \to \mathbb{N}_c^+ \; \forall x, x', x'' \in \mathbb{N}_k^+ \; (f(x) = f(x') = f(x'') \Rightarrow x \neq x' + x''\}.$$

That this is a good definition, *i.e.* that $\mathbf{h}(c)$ is a finite integer for every $c > 0$, is a consequence of Schur's theorem (a corolary of Ramsey Theorem).

Trivially $\mathbf{h}(1) = 3$:

$\mathbf{C_1}$
1
2

In the same way, it is easy to verify that $\mathbf{h}(2) = 9$. The following is the maximal solution for $c = 2$:

$\mathbf{C_1}$	$\mathbf{C_2}$
1	3
2	5
4	6
8	7

It was proven that for $c = 3$, 24 is the length of the maximal game. An example of a solution of that length is

$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$
1	3	9
2	5	10
4	6	12
8	7	13
11	19	14
16	21	15
22	23	17
		18
		20

For the case $c = 4$, Harary announced that he knew a solution of length 66, generated by a computer program, but he didn't know if it was maximal.

2 Taming brute-force

A brute-force approach to the problem is very easy to state: a program that tries every possible different choice of placement of each integer, with a depth-first search for example, thus covering all possible game configurations. The longest one is the answer to our question. The problem is, that for the game with 4 columns it simply **takes to much time to do this!**

It is easy to see that the whole game configuration can be represented by the sequence of columns chosen in each turn by the players to place the numbers, as those numbers are determined by the natural order of $\mathbb{N}$. With this observation the maximal game for the problem $c = 3$ can be completely described by the string:

$$[C_1, C_1, C_2, C_1, C_2, C_2, C_2, C_1, C_3, C_3, C_1, C_3, C_3, C_3, C_3, C_1, C_3, C_3, C_2, C_3, C_2, C_1, C_2]$$

or, in a lighter notation, by

$$[1, 1, 2, 1, 2, 2, 2, 1, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 2, 3, 2, 1, 2].$$

If we, as usual, denote $A^\star$ as the Kleene closure of A, game descriptions can be seen as a member of $(\mathbb{N}_c^+)^\star$, that is, the set of sequences of elements of $\mathbb{N}_c^+$. To simplify the representation, and because we will need computationally fast implementations of these

data structures, let us assume we have an upper bound to the length of the maximal game for a given c. Let us represent that limit by `LIMIT`. Then, if we concatenate a string of 0's to a string description of a game, so that the result has always length `LIMIT`, we can see a game description as a member of

$$(\mathbb{N}_c)^{\texttt{LIMIT}}.$$

This can be easily and efficiently represented in a computer data structure.

In this case, brute-force means to generate all the possible elements of

$$(\mathbb{N}_c)^{\texttt{LIMIT}} \cap (\mathbb{N}_c^+)^\star\{0\}^\star$$

that stand for legal configurations of the game. So we only need to execute the following code:

```
#define NCOLS 4
#define LIMIT 70

void brute_force(){
  int i;

  for(i=0; i<NCOLS; i++)
    if(column_selectable(i)){
      select_column(i);
      brute_force();
      backtrack_last_move();
    }
}

main(){
   init_data_structures();
   brute_force();
}
```

The problem is the computational that the cost of `column_selectable()` is too high. A direct evaluation from the data of the game configuration, for the nth placement will have to compute in the worst case $(n-1)(n-2)$ additions and comparisons, for a configuration with all previous placements in the same column. We can estimate an average of $c(\frac{n}{c})^2$, supposing an even distribution of placements by the different columns. This is clearly too much for a computation that, in the case $c = 4$, is going to be repeated a number of times that we only know to bounded by 4^{66}.[1]

A solution for this problem is to use another data structure, that can store the "forbidden values" for each column, and that can be calculated in an incremental way. For each column we can store the values that can be obtained by adding two of the members of the column. Lets call that structure `base` and use it so that `base[i][j] == 0` means that the number `j` can be "legally" placed in column `i+1`.

```
#define column_selectable(X) (base[X][last] == 0)
```

[1] Assuming that the solution already known for the problem, is indeed maximal.

```
int base[NCOLS][LIMIT+1], sol[LIMIT], last=0;

void init_data_structures(){
  int i,j;

  for(i=0; i<NCOLS; i++)
    for(j=0; j<=LIMIT; j++)
      base[i][j] = 0;
}

void select_column(int col){
  int i, sum;

  sol[++last] = col;
  for(i=1; i<last; i++){
    sum = last+i;
    if((sol[i]==col) && (sum <= LIMIT))
      base[col][sum]++;
  }
}
```

Testing if a move is legal is now much more efficient ($O(n)$) but the backtracking of a move, will still take too much time. We are already storing in `base[col][i]` not simply a 1 when the integer `i` is "forbidden" on that `col`, but the number of different ways it is possible to write it as sum of two elements of the column. In this way backtracking will be easier when one of the contributing parcels of `i` is removed from column `col` it needs just to subtract 1 to this value (`base[col][i]--;`). If we keep, in another data array, for each integer `i`, the list of integers than it contributes to "outcast" in its column, backtracking can be done with minimal computational effort ($O(n)$). Rewriting the last code to incorporate these speed-ups, we have:

```
#define column_selectable(X) (base[X][last] == 0)

int base[NCOLS][LIMIT+1], sol[LIMIT], last;
int sums[LIMIT+1][LIMIT+1];

void init_data_structures(){
  int i,j;

  for(i=0; i<NCOLS; i++)
    for(j=0; j<=LIMIT; j++)
      base[i][j] = 0;
}

void select_column(int col){
  int i, j=0, sum;

  sol[++last] = col;
  for(i=1; i<last; i++){
```

```
    sum = last+i;
    if((sol[i]==col) && (sum <= LIMIT)){
      sums[last][j++]=sum;
      base[col][sum]++;
  }
}

void backtrack_last_move(void){
  int i=0, col;

  col = sol[last];
  while(sum[last][i]){
    base[col][sums[last][i]]--;
    sums[last][i++] = 0;
  }
  last--;
}
```

3 Trimming the tree

We can significantly improve computational speed by simply not doing so much, *i.e.* pruning the search tree whenever we know that, although the search will find new maximal solutions, they will be of the same length of the ones already found. Solutions that can be obtained from others by renaming (or reorder if you prefer!) are not of particular interest.

So, we can fix the first column to be chosen, say the first, and save time reducing it to $\frac{1}{c}$ (in this case $\frac{1}{4}$). But other, and much more effective pruning criteria can be used.

Let us consider again our domain

$$(\mathbb{N}_c)^{\texttt{LIMIT}} \cap (\mathbb{N}_c^+)^\star\{0\}^\star.$$

The depth-first descend that we are executing corresponds to the enumeration in lexicographic order. Lets denote by $[a/b]s$ the application of the substitution of every b by a in the string s. Then we observe that for any string

$$\mathbf{s} = a_1 a_2 \dots a_{n-1} a_n a_{n+1} \dots a_k$$

if $a_n > (max\{a_1 \dots a_{n-1}\} + 1)$,

$$\mathbf{s}' < \mathbf{s}$$

where

$$s' = [(max\{a_1 \dots a_{n-1}\} + 1)/a_n, a_n/(max\{a_1 \dots a_{n-1}\} + 1)]s.$$

The placement of number 1 in the first column, can be seen as a special case of this last observation. The `main()` and `brute_force()` functions can then be rewritten as:

```
#define MAX(X,Y) (X<Y?Y:X)
#define MIN(X,Y) (X<Y?X:Y)

void brute_force(int bound){
  int i;
```

```
  for(i=0; i<=(MIN(NCOLS-1,bound+1)); i++)
    if(column_selectable(i)){
      select_column(i);
      brute_force(MAX(i,bound));
      backtrack_last_move();
    }
}

main(){
  init_data_structures();
  brute_force(-1);
}
```

If we take LIMIT = 4 and $c = 4$, only the following 15 complete configurations will be tested for validity, instead of the total possible 256:

1	1	1	1
1	1	1	2
1	1	2	1
1	1	2	2
1	1	2	3
1	2	1	1
1	2	1	2
1	2	1	3
1	2	2	1
1	2	2	2
1	2	2	3
1	2	3	1
1	2	3	2
1	2	3	3
1	2	3	4

For each c, if we consider the language

$$L_c = \{a_1 a_2 \dots a_k \in (\mathbb{N}_c^+)^\star \mid \forall i \in \mathbb{N}_k^+, a_i \leq max\{a_1, \dots, a_{i-1}\} + 1\}$$

we can restrict the domain of brute force search to

$$(\{0\} \cup \mathbb{N}_c^+)^{\texttt{LIMIT}} \cap L_c\{0\}^\star$$

The languages L_c are regular languages, as they can be described by the regular expressions

$$\alpha_c = \bigoplus_{i=1}^{c} \bigotimes_{j=1}^{i} j(1 + \ldots + j)^\star$$

From these expressions it is easy to get for each n the density of L_c, $\rho_{L_c}(n)$, i.e, the number of strings of length n that are in L_c:

$$\rho_{L_c}(n) = \sum_{i=1}^{c} S(n, i)$$

where

$$S(n,i) = \frac{1}{i!}\sum_{j=0}^{i-1}(-1)^j \binom{i}{j}(i-j)^n$$

are the Stirling numbers of second kind. And we have

$$\rho_{L_c\{0\}^\star}(n) = \sum_{k=0}^{n}\sum_{i=1}^{c} S(k,i)$$

For $c = 4$, we obtain,

$$\rho_{L_4}(n) = \frac{1}{3} + \frac{1}{4}\,2^n + \frac{1}{24}\,4^n$$

and

$$\rho_{L_4\{0\}^\star}(n) = \frac{1}{3}n - \frac{5}{9} + \frac{1}{2}\,2^n + \frac{1}{18}\,4^n$$

Finally, the density of $L_4\{0\}^\star$ can be compared with that of $(\mathbb{N}_4^+)^\star\{0\}^\star$, noticing that

$$\rho_{(\mathbb{N}_4^+)^\star\{0\}^\star}(n) = \frac{4}{3}\,4^n - \frac{1}{3}$$

For $n = 70$, $\rho_{L_4\{0\}^\star}(70) \approx 2^{135.8}$ and $\rho_{(\mathbb{N}_4^+)^\star\{0\}^\star}(70) \approx 2^{140.4}$, which shows that we have cut the tree in a factor of about 24 (i.e $\approx \frac{1}{c!}$).

4 The result

Running the program for $c = 4$ took 333501s (less than 4 days) in a *Intel Pentium 4* with a clock rate of 3.0GHz and listed the 29931 maximal game configurations with length 66.

We will not list the complete set of configurations obtained (!), but can here present the first of the long list of solutions:

[1121222133133331332321241444441444221441441444444122233313313312222]

And thus we have confirmed that indeed $\mathbf{h}(4) = 67$.

For $c = 3$, after less than 0.01s all the maximal configurations of length 24 were listed:

[112122213313333133232121]
[112122213313333313232121]
[112122213313333333232121]

5 Some variations

If we try some variations on the rules of the game, and instead of those stated in section 1 we say that **an integer can be placed in a column providing it cannot be written as the sum of elements of that column**, *i.e.* allowing sums with more than exactly two parcels, we have a completely different problem. All the structures and code written before can be used, providing we have new definitions for `select_column()` and `backtrack_last_move()`:

```
void select_column(int col){
  int i, j=0, sum;
```

```
  sol[++last] = col;
  for(i=LIMIT; i>=1; i--)
    if(base[col][i]){
      sum = last + i;
      if(sum <= LIMIT){
        base[col][sum]++;
        sums[last][j++] = sum;
      }
    }
  base[col][last]++;
}

void backtrack_last_move(void){
  int i=1, col;

  col = sol[last];
  while(sums[last][i]){
    base[col][sums[last][i]]--;
    sums[last][i++] = 0;
  }
  base[col][last--]--;
}
```

The complexity is now much lower than in the initial problem, as the condition for a placement to be admissible is much more strict[2]. The answers for $c = 1, 2, 3, 4$ are as follow:

$\mathbf{h}'(1) = \mathbf{h}(1) = 3$ The only possible game configuration is obviously

$$[11]$$

$\mathbf{h}'(2) = \mathbf{h}(2) = 9$ The only maximal configuration is the first that was found for the original problem,

$$[11212221]$$

$\mathbf{h}'(3) = 22$ The only maximal configuration is

$$[112122213333333333221]$$

$\mathbf{h}'(4) = 47$ The five maximal game configurations are

$$[1121222133333333331242444444444444444444444212]$$
$$[1121222133333333331244444444444444444444441222]$$
$$[1121222133333333331244444444444444444444442221]$$
$$[1121222133333333332144444444444444444444441222]$$
$$[1121222133333333332144444444444444444444442221]$$

6 Conclusion

We described a brute-force algorithm used for determining the value of $\mathbf{h}(4)$. Adequate data-structures allowed to test a legal move and backtracking with an $O(n)$ time complexity. We shown how to prune the search tree, maintaining completeness, and represented

[2]The answer for the problem with $c = 4$ was given after using only 98.37s in the same P4 3GHz.

recurrence relations for the estimate space. The same strategies were used for solving some variations of the initial problem. We would like to study winning strategies for each number of columns.

Acknowledgement

We thank the Parallel and Distributed Systems Group of LIACC, and specially Luís Lopes, for the uncluttered CPU-time in *khuasar*.