Paper without a name

A. Auvolat A. Fromherz N. Jeannerod

June 01, 2014

Abstract

In this contribution, we investigate the problem of "SwitchBoxes". The goal is, given n wires, to generate all the permutations of these wires by using "boxes", that swap two wires. We tried to minimize the number of boxes that were necessary. We introduce a conjecture that puts our problem in relation with binary insertion sort. We successfully used a heavily optimized algorithm to prove this conjecture for small values of n.

1 Introduction

The SwitchBoxes problem is a combinatory problem. Given n wires, we try to generate all the permutations of these wires by using boxes. A box takes two wires and a control bit, and swaps the two entry wires iff the control bit is set. We can concatenate these boxes to get a configuration of boxes. The configuration is said valid if, by giving the set of boxes different configurations (ie. swapping or not the wires on every box), we can obtain any permutation of the n wires.

Mathematically, we consider that the wires are numbers from 1 to n. A box is a permutation $\tau(\epsilon) = (i, j)^\epsilon$, where ϵ represents the control bit for the box and is either 0 or 1. A configuration of boxes C is the concatenation of k boxes $\tau_1, \ldots, \tau_k$, that means, $C(\epsilon_1, \ldots, \epsilon_k) = \tau_1^{\epsilon_1} \circ \ldots \circ \tau_k^{\epsilon_k}$. The configuration C is said valid if $C(\{0, 1\}^k) = \mathfrak{S}_n$.

Given this problem, we tried to determine the best lower bound of the number of boxes k which is valid for $\mathfrak{S}_n$, ie generates all the permutations of n wires.

We can easily determine trivial lower and upper bounds of the number of boxes :
Given the fact that C is an application, we need to have $Card(\{0; 1\}^k) \geq Card(\mathfrak{S}_n)$. We immediately obtain that k has to be greater than $\log_2(n!)$.
We also have a reachable upper bound : The traditional bubblesort gives a valid configuration of the SwitchBoxes problem. We obtain that the optimal k is smaller than $\frac{(n-1)(n-2)}{2}$.

In this contribution, we first recall how the sequence of the maximal number of comparisons for sorting n elements by binary insertion is defined and develop how the optimal k seems to be linked to it. We then explain why we guess that those numbers are equal.

In a second step, we show how we experimentally checked this conjecture for n from 2 to 6, and how we demonstrated that the number of comparisons is an upper bound of the optimal

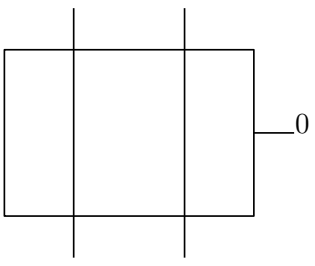

Figure 1: A switch box with control bit not set

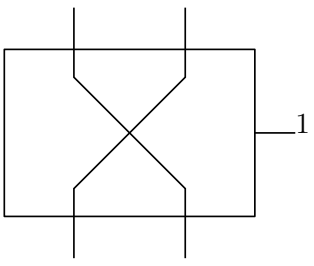

Figure 2: A switch box with control bit set

number of boxes for n from 7 to 13 by optimizing the checking algorithm and by constructing a precise configuration.

In addition, we give some ideas we tried to develop in order to prove mathematically this conjecture.

In the rest of this paper, opt_n will be the optimal number of boxes for n wires.

2 The A001855 sequence

In this paragraph, we will develop on the A001855 sequence, here denoted u_n, which seems to be in relation with our problem.

u_n is defined as the maximal number of comparisons done by a binary insertion sort running on a table of n items.

Algorithm 1 Binary insertion sort algorithm

```
function FindPos(n, T, b, e)                    ▷ Find n in T[b, b+1, ..., e-1]
    assert b < e
    if b + 1 = e then
        return b
    else
        m ← ⌊(b + e)/2⌋
        if T[m] > n then
            return FindPos(n, T, b, m)
        else
            return FindPos(n, T, m + 1, e)
        end if
    end if
end function

function Sort(T)
    n ← size(T)
    S ← [T[0]]
    for i = 1 to n − 1 do
        p ← FindPos(T[i], S, 0, size(S))
        insert T[i] in S at p
    end for
    return S
end function
```

This algorithm progressively builds S, the table of the elements of T in increasing order. This is done by progressively inserting the elements of T in S at a position calculated by the *FindPos* function. The *FindPos* function is based on a binary search, therefore does $O(log(\text{size}(S)))$ comparisons.

The u_n suite is defined as follows :

$$\begin{aligned} u_1 &= 0 \\ \forall n \in \mathbb{N}^*, u_{n+1} &= u_n + \lceil log_2(n) \rceil \end{aligned}$$

This sequence is described on the OEIS[1], and is referenced under the name A001855.

This sequence is linked to the construction of complete binary trees : u_n is the sum of the depths of all nodes in the complete binary tree containing $n-1$ nodes (see [**?**]).

[1] Online Encyclopedia of Integer Sequence

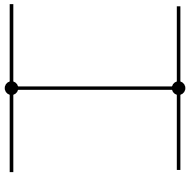

Figure 3: Trivial solution for 2 wires

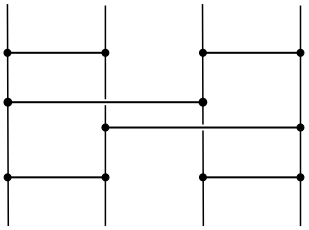

Figure 4: Simple non-optimal solution for 4 wires

Number of wires n	opt_n
2	1
3	3
4	5
5	8
6	11

Figure 5: Minimum box count for small number of wires, checked experimentally

3 Our conjecture

To understand the problem, we first tried manually to get opt_n for very small values of n. For $n = 2$, the answer is quite simple : One box is enough and necessary to swap the two wires.

For $n = 3$, two boxes aren't enough. Using a permutation, we can consider that the first box swap the wires 1 and 2, and that the second box swap the wires 2 and 3. Then the permutation $(1, 3)$ can't be generated with this configuration. Hence, we have $opt_3 = 3$.

On these representations, a box is a horizontal line that can swap the wires the points are on.

For greater values of n, we implemented a naive and brutal algorithm that for k given, creates all the possible configurations of k boxes, and checks if one of these configurations is valid. We first try it with $k = opt_{n-1} + 1$. Then, we increment k and try again until we get a valid configuration. With this algorithm, we easily get opt_n.
However, this only works for small values of n. For n greater than 6, we lack memory and time to achieve the computation.

Using this algorithm, we got the results in the table of figure **??**.

According to the fact that this sequence is exactly the beginning of the u_n sequence described in the previous paragraph, our conjecture is that the sequences are equal.

Conjecture. *opt_n is equal to the maximal number of comparisons for sorting n elements by binary insertion.*

In order to prove this conjecture for greater values, we tried to optimize our checking algorithm. That will be described in the next part.

3.1 First examples

$n = 2 : C_2(e_1) = (1, 2)^{e_1}$
$n = 3 : C_3(e_1, e_2, e_3) = (1, 2)^{e_1} \circ (2, 3)^{e_2} \circ (1, 2)^{e_3}$
$n = 4 : C_4(e_1, e_2, e_3, e_4, e_5) = (1, 2)^{e_1} \circ (3, 4)^{e_2} \circ (1, 3)^{e_3} \circ (2, 4)^{e_4} \circ (1, 2)^{e_5}$

3.2 Notation

We abbreviate the previous notation in the following way :

$$C_2 = (1, 2)$$

$$C_3 = (1, 2)(2, 3)(1, 2)$$

$$C_4 = (1, 2)(3, 4)(1, 3)(2, 4)(1, 2)$$

4 Hypothesis box system

We have found a box system that seems to generate all permutations for all n. We have managed to prove that this box system is valid for small values of n ($n \leq 13$), and have found that the box systems were still valid when some boxes were removed.

4.1 Explaining our intuition

We work here with $n = 2^p$ wires. Our idea is that we need to be able to route any of the n input wires to any of the n output wires. A box enables the *swapping* or *non-swapping* of two wires. Its behaviour is tightly linked with that of the two wires.

We tried to imagine a step of *multiplexing* followed by a step of *demultiplexing*, that would regroup all the wires in one point before deciding which wire goes where. Since one box is plugged on two wires, we can imagine its output as being one bus of two wires. The idea is thus relatively simple :

- We begin with $n = 2^p$ simple wires
- We group them two by two into 2^{p-1} buses of size 2
- We group the buses two by two into 2^{p-1} buses of size 4
- ...
- We group the two buses of size 2^{p-1} into one bus of size 2^p.

Now the demultiplexing step is just the opposite :

- We begin with one bus of 2^p wires that we separate into two smaller buses of 2^{p-1} wires
- ...
- We separate the 2^{p-1} buses of 2 wires into 2^p simple wires.

The grouping/separation of wires requires a given number of boxes, and we tried to put just that number of boxes in the box system we designed. Grouping two buses of k wires requires k boxes.

We tried this box system and found, for small values of n, that it worked.

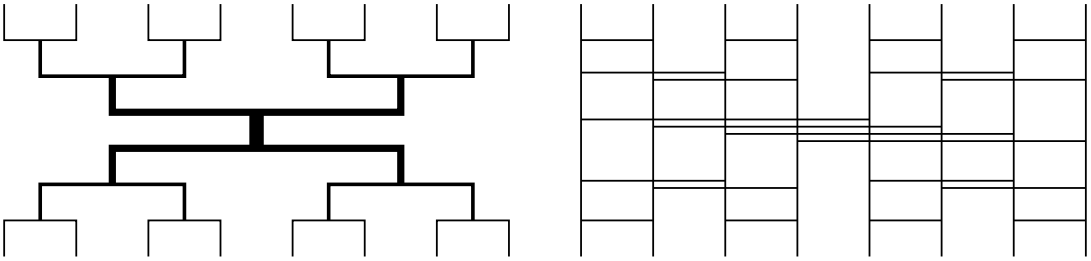

Figure 6: Explanation of our intuition leading to the construction of $C^0_{2^j}$

4.1.1 Number of boxes

Let $n = 2^p$ be the number of wires. On each line of grouping/separating we have $n/2$ boxes. We now need to count lines of grouping and of separating. We begin from $n = 2^p$ wires and want to group them into 1 bus, therefore we need p lines of grouping. Conversely, we then need to separate one bus into 2^p wires, therefore requiring p lines of separating.

We notice that the last line of grouping and the first line of separating have exactly the same boxes and each wires is used by exactly one grouping box and one separating box. These two lines are thus redundant and we only put one of them.

Finally :

$$\begin{aligned} \#\text{boxes} &= \#\text{boxesperline} \times (\#\text{grouplines} + \#\text{separatelines} - 1) \\ &= \tfrac{n}{2}(2p-1) \\ &= n^{p-1/2} \\ &= n^{log_2 n - 1/2} \end{aligned}$$

4.2 The basis box configuration

We first define $B(q,i)$, the q-block at position i by :

Définition.

$$\begin{aligned} B(q,i) &= (q, q+2^i)(q+1, q+1+2^i)\cdots(q+2^i-1, q+2*2^i-1) \\ &= \bigcirc_{j=0}^{2^i-1} (q+j, q+j+2^i) \end{aligned}$$

For example:

$$\begin{aligned} B(q,0) &= (q, q+1) \\ B(q,1) &= (q, q+2)(q+1, q+3) \\ B(q,2) &= (q, q+4)(q+1, q+5)(q+2, q+6)(q+3, q+7) \end{aligned}$$

Then, for $n = 2^p$ we define $L(p,q)$, for $q < p$ the (p,q)-line by :

Définition. $L(p,q) = B(q,0)B(q,2^{q+1})\cdots B(q,2^p - 2^{q+1})$

A line is just the union of disjoint blocks of same size, so that all the wires are used by the blocks.

For example:

$$L(3,0) = (0,1)(2,3)(4,5)(6,7)$$
$$L(3,1) = (0,2)(1,3)(4,6)(5,7)$$
$$L(3,2) = (0,4)(1,5)(2,6)(3,7)$$

We now conjecture that the following configuration generates all permutations for $n = 2^p$:

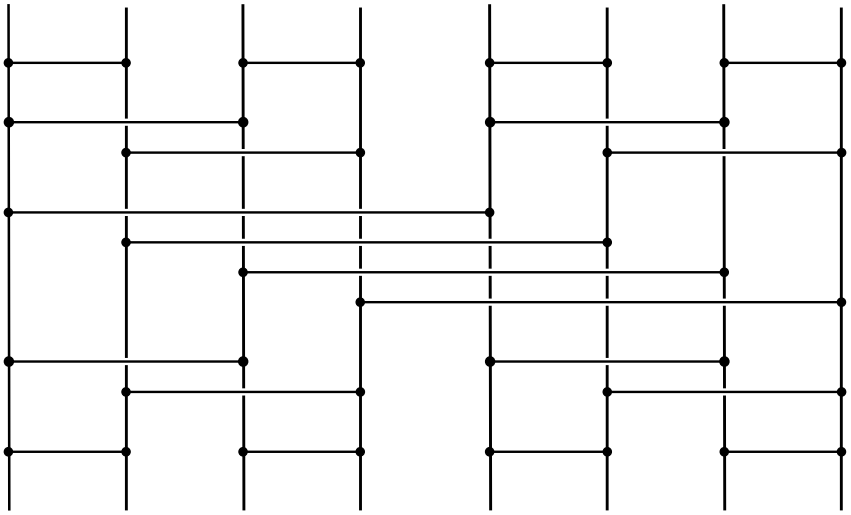

Figure 7: Representation of C_8^0

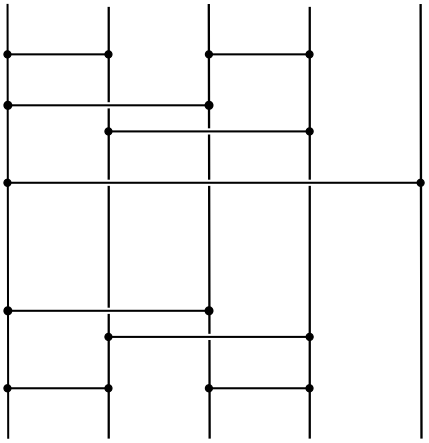

Figure 8: Solution for 5 wires: C_5^0

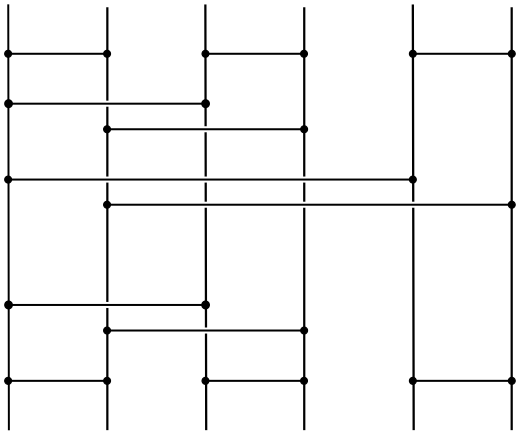

Figure 9: Solution for 6 wires: C_6^0

Définition. $C_{2^p}^0 = L(p,0)L(p,1)L(p,2)\cdots L(p,p-1)L(p,p-2)\cdots L(p,1)L(p,0)$

For example :

$$\begin{aligned} C_8^0 &= L(3,0)L(3,1)L(3,2)L(3,1)L(3,0) \\ &= (0,1)(2,3)(4,5)(6,7) \\ &\quad (0,2)(1,3)(4,6)(5,7) \\ &\quad (0,4)(1,5)(2,6)(3,7) \\ &\quad (0,2)(1,3)(4,6)(5,7) \\ &\quad (0,1)(2,3)(4,5)(6,7) \end{aligned}$$

4.3 Downscaling

For n which is not a power of two, we construct C_n^0 by taking $C_{2^{\lceil log_2(n) \rceil}}^0$ and keeping only boxes (i,j) having $i,j < n$.

For example, C_5^0 can be generated from C_8^0 and by removing 11 of the 20 boxes. We obtain the following solution :

$$C_5^0 = (0,1)(2,3)(0,2)(1,3)(0,4)(0,2)(1,3)(0,1)(2,3)$$

Conjecture. *C_n^0 generates all permutations fon n*

Number of wires n	Size of C_n^0	Size of C_n^1	opt_n	u_n
2	1	1	1	1
3	3	3	3	3
4	6	5	5	5
5	9	8	8	8
6	12	11	11	11
7	15	14		14
8	20	17		17
9	25	21		21
10	28	25		25
11	31	29		29
12	34	33		33
13	39	37		37

Figure 10: Minimum box count for small number of wires, checked experimentally

4.4 Testing the box system

For a box system $C = b_1 b_2 \dots b_p$, we construct the sets :

$$Perm_k = (b_1 b_2 \dots b_k)(\{0,1\}^k)$$

These sets are recursively defined by :

$$\begin{array}{lcl} Perm_0 & = & id \\ Perm_{k+1} & = & Perm_k \cup \{\sigma \circ b_{k+1},\ \sigma \in Perm_k\} \end{array}$$

The permutation set $Perm_p$ therefore contains all the possible permutations we can generate with the p boxes. If $Perm_p = \mathfrak{S}_n$ then the box system is valid.

Each set $Perm_k$ is represented in memory as a bitset containing $n!$ bits, one for each permutation of $\mathfrak{S}_n$. The isomorphism between the permutations of $\mathfrak{S}_n$ and the numbers $\{0, 1, \dots, n! - 1\}$ is constructed using the standard Lehmer code.

The representation as a bitset is the most compact representation we have found.

4.5 Useless box elimination

We can easily detect that $Perm_{k+1} = Perm_k$ when $|Perm_{k+1}| = |Perm_k|$, because $Perm_k \subset Perm_{k+1}$. When this condition is fulfilled, then we know that the box b_{k+1} is useless because it adds no permutation to the set of generated permutations. We can therefore remove it from the box system.

By removing the boxes that this method detects as useless from the box system we constructed in the previous paragraphs, we have constructed box systems that generate all the permutations for $n \leq 13$ and that have u_n boxes (ie the conjectured optimum). We note the simplified system for n wires C_n^1. We have not managed to run the test for $n \geq 14$, because the required RAM exceeds 20Go, which is not a resource we have in our possession.

For example for $n = 5$, we have :

$$C_5^0 = (0,1)(2,3)(0,2)(1,3)(0,4)(0,2)(1,3)(0,1)(2,3)$$

$$C_5^1 = (0,1)(2,3)(0,2)(1,3)(0,4)(0,2)(0,1)(2,3)$$

5 Conclusion

All these results seem to confirm our intuition. To go further, we unsuccessfully tried to find applications from the set of trees representing a sort by binary insertion to the set of valid configurations of boxes that would be either one to one or onto, in order to find an inequality between these two sequences of integers. We also tried to put our construction of configurations in relation with binary insertion sort.

References

[1] Sung-Hyuk Cha, *On Integer Sequences Derived from Balanced k-ary Trees*, Applied Mathematics in Electrical and Computer Engineering, 2012.