Mixed Algorithm for Combinations Generation

Constantin Ciubotaru

Abstract

Modifications are proposed to the recursive algorithm of combinations generation by reducing the number of non-performing recursive calls and recovering this gap through iterative processes.
Keywords: combinations, combinations generation, recursive algorithm, iterative algorithm, `Common LISP`, `Rosetta Code`.

1 Introduction

To calculate the number of all k-combinations from a given set of n elements ($0 \leq k \leq n$) we will use the well known formula

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

However, not only the number of combinations is often important, but also generation (enumerating) of all possible combinations, e.g., for some optimization problems solution. In this case we will use the formula below. It can easily be deduced:

$$C_{n-1}^k = \frac{(n-1)!}{k!(n-k-1)!} = \frac{(n-1)!(n-k)}{k(k-1)!(n-k-1)!(n-k)} = \frac{n-k}{k} C_{n-1}^{k-1}.$$

This equality helps us to demonstrate the following recurring formula:

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k. \tag{1}$$

Really

$$C_{n-1}^{k-1} + C_{n-1}^k = C_{n-1}^{k-1} + \frac{n-k}{k} C_{n-1}^{k-1} = (1 + \frac{n-k}{k}) C_{n-1}^{k-1} =$$

$$= \frac{n(n-1)!}{k(k-1)!(n-k)!} = \frac{n!}{k!(n-k)!} = C_n^k.$$

Formula (1) can also be transcribed in the following way:

$$C_n^k = C_{n-1}^{k-1} + C_{n-2}^{k-1} + \ldots + C_k^{k-1} + C_k^k. \tag{2}$$

Formulas (1) and (2) constitute the basis of recursive algorithms for combinations generation. Of course, the elegance and the way of implementing the recursive algorithms by stacks impresses. In some situations, however, recursive calls become too expensive. The efforts made to organize the recursive call outweigh the calculations scheduled within the call. In such situations it is more efficient to substitute recursion by iteration. The difficulty that appears here is to determine the boundary of separation between recursive process and the iterative one. Just this is the case of combinations generation. We will propose some modifications to the recursive algorithm by inserting iterative elements.

2 Recursive Algorithm for Combination Generation

Let M be an arbitrary set of n elements, and *com* – an arbitrary k-combination of distinct elements from M or *nil* (empty combination, notation from `Common LISP`). For example, M={1,2,3}, *com* = (13), $M = \{a, b, c, d\}$, *com*=(*abd*). If we denote $C_M^k = \{c_1, c_2, \ldots, c_m\}$, then $C_M^k \cdot com = \{c_1 \cdot com, c_2 \cdot com, \ldots, c_m \cdot com\}$. The concatenation operation is denoted by "$\cdot$". For example, $C_{\{1,2,3\}}^2 = \{(12), (13), (23)\}$, $C_{\{1,2,3\}}^2 \cdot (45) = \{(1245), (1345), (2345)\}$, and $C_{\{1,2,3\}}^2 \cdot nil = \{(12), (13), (23)\}$. It should be noted that the order of elements in combination is not relevant.

Let us denote further $M = \{1, 2, ..., n\} = M_1$, $M_2 = \{2, 3, ..., n\}$, $M_3 = \{3, 4, ..., n\}$,..., $M_n = \{n\}$. Using these denotations and Formula (2) we define C_M^k:

$$C_M^k = C_{M_2}^{k-1} \cdot (1) \cup C_{M_3}^{k-1} \cdot (2) \cup ... \cup C_{M_{n-k+1}}^{k-1} \cdot (n-k) \cup C_{M_{n-k+1}}^k. \tag{3}$$

Let us make sure that this formula really computes all possible k-combinations over the M. First, let us define $C_M^0 = \{nil\}$ for any M, including $M = \varnothing = \{\}$ and $C_M^k = \{(i_1 i_2 \dots i_k)\}$ for $M = \{i_1, i_2, \dots, i_k\}$, $k = card(M)$.

Theorem

The set C_M^k contains all possible $C_{card(M)}^k$ k-combinations of elements from the set M, $0 \leq k \leq card(M)$.

Proof

Let $M = \{1, 2, ..., n\}$. We will prove that the theorem is true for any C_M^i by induction on i. For $i=0$, by definition, $C_M^0 = \{nil\}$, $C_{card(M)}^0 = 1$. For $i = 1$ we obtain: $C_M^1 = C_{M_2}^0 \cdot (1) \cup C_{M_3}^0 \cdot (2) \cup ... \cup C_{M_n}^0 \cdot (n-1) \cup C_{M_n}^1 = \{nil\} \cdot (1) \cup \{nil\} \cdot (2) \cup ... \cup \{nil\} \cdot (n-1) \cup \{(n)\} = \{(1), (2), ..., (n-1), (n)\}$.

Let's assume that the statement is true for $i = 0, 1, ..., k$, and we will prove it for $i = k+1$. So, C_M^i contains all possible i-combinations of elements from M, in total $C_{card(M)}^i$ combinations, $i = 0, 1, ..., k$. We will show that C_M^{k+1} contains $C_{card(M)}^{k+1}$ distinct $(k+1)$-combinations of elements from M. According to the Formula 3

$$C_M^{k+1} = C_{M_2}^k \cdot (1) \cup C_{M_3}^k \cdot (2) \cup ... \cup C_{M_{n-k}}^k \cdot (n-k-1) \cup C_{M_{n-k}}^{k+1}. \quad (4)$$

Note that:

- $card(M_{n-k+1}) = k+1$, so, $C_{M_{n-k+1}}^{k+1} = \{(n-k+1\, n-k\, ...n-1\, n)\}$, $C_{k+1}^{k+1} = 1$.
- $C_{M_i}^k \cdot (i-1) \cap C_{M_j}^k \cdot (j-1) = \varnothing$ for any $i \neq j$. The scheme of possible layout of the sets M_i, M_j is shown in Figure 1. In the case a) $(i-1) \notin M_j$, and in the case b) – $(j-1) \notin M_i$.
- $C_{M_i}^k \cdot (i-1) \cap C_{M_{n-k+1}}^{k+1} = \varnothing$ for all $2 \leq i \leq n-k+1$. The case c) is clear from Figure 1.
- $card(C_{M_i}^k \cdot (i-1)) = card(C_{M_i}^k) = C_{card(M_i)}^k$.

So we get:

$$card(C_M^{k+1}) = C_{n-1}^k + C_{n-2}^k + ... + C_{k+1}^k + C_{k+1}^{k+1} = C_{card(M)}^{k+1}. \quad (5)$$

(Formula 2).

To complete the proof, let us mention that all combinations in $C^k_{M_i} \cdot (i-1)$ are distinct $(k+1)$-combinations. ■

Further we will refer to the recursive algorithm for combinations generation written in the `Common LISP` language [1,2] and published on the portal `Rosetta Code` [1] [3] (Figure 2). The termination conditions

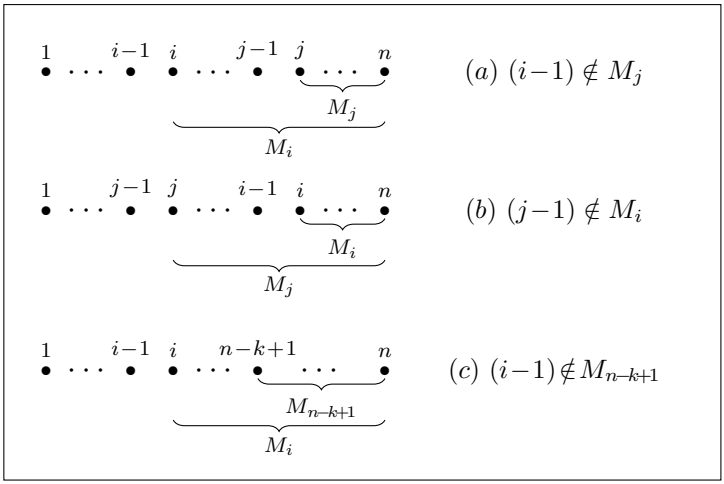

Figure 1: Schematic placement of M_i, M_j, M_{n-k+1}

for recursive calls in `Rosetta COMB` using the above denotations are: 1) $C^k_M \cdot com = C^0_M \cdot com$ $(k{=}0)$ and 2) $C^k_M \cdot com$, $card(M) < k$.

In the first case the function will return the *com* value, and in the second case we have a deadlock, that is, the respective call does not generate anything. It can be noticed that for $k{=}0$, *com* will be exactly one possible k-combination.

[1] Rosetta Code is a wiki-based programming chrestomathy website with implementations of common algorithms and solutions to various programming problems in many different programming languages.

Rosetta COMB

```
(defun comb (k lst)
  (labels((comb1 (l c k)
        (when (>= (length l) k)
              (if (zerop k) (return-from comb1 (print c)))
              (comb1 (cdr l) c k)
              (comb1 (cdr l) (cons (first l) c) (- k 1)))))
      (comb1 lst nil k)))

; (comb 3 '(1 2 3 4 5))

; (5 4 3) (5 4 2) (5 3 2) (4 3 2) (5 4 1) (5 3 1) (4 3 1) (5 2 1) (4 2 1) (3 2 1)
```

Figure 2: The recursive function `Rosetta COMB` for combinations generation.

3 Modified Algorithm for Combination Generation

We will modify the `Rosetta COMB` function (Figure 2) by reducing the number of non-performing recursive calls and recovering this gap through iterative processes. For this purpose we change the termination conditions for recursive calls. The new conditions that ensure the convergence of the recursive process and decrease the total number of calls are:

$$1)\ C_M^1 \cdot com\ (k=1) \text{ and } 2)\ C_M^{card(M)} \cdot com\ \ (k{=}card(M)).$$

In the first case all 1-combinations of elements from M concatenated with com will be generated, in total $card(M)$ combinations, and in the second case one single k-combination will be generated, $k = card(M)$. Namely these conditions generate iterative processes. The function so modified is shown in Figure 3.

For C_5^3 the modified function produces 11 recursive calls, and the initial function – 69 calls. For C_{12}^5 these indicators are 659 and 2573 respectively, and for C_{26}^{13} – 10400599 and 40116599, respectively. The

modified function recovers this gap by calling (iteratively) the `dolist` and `append` functions.

COMB Modified

```
(defun comb (k lst)
  (labels((com (l c k)
    (cond((= (length l) k)(print(append l c)))
         ((>= (length l) k)
         (cond((= 1 k) (dolist (x l) (print(cons x c))))
              (t(com (cdr l) (cons(car l) c)(- k 1))(com (cdr l) c k)))))))
     (com lst nil k)))

; (comb 3 '(1 2 3 4 5))

; (3 2 1) (4 2 1) (5 2 1) (4 3 1) (5 3 1) (4 5 1) (4 3 2) (5 3 2) (4 5 2) (3 4 5)
```

Figure 3: The recursive function `Rosetta COMB Modified` for combinations generation.

To collect all combinations, we can use an auxiliary parameter, replacing the `print` functions with collection functions, for example, `push` (Figure 4).

COMB Modified with collection

```
(defun comb (k lst &aux rez))
  (labels((com (l c k)
    (cond((= (length l) k)(push(append l c)rez))
      ((>= (length l) k)
      (cond((= 1 k) (dolist (x l) (push(cons x c)rez)))
        (t(com (cdr l) (cons(car l) c)(- k 1))(com (cdr l) c k)))))))
(com lst nil k))rez)

; (comb 3 '(1 2 3 4 5))
;((3 4 5) (4 5 2) (5 3 2) (4 3 2) (4 5 1) (5 3 1) (4 3 1) (5 2 1) (4 2 1) (3 2
    1))
```

Figure 4: The recursive function `COMB Modified` with the collection of results.

Using the primitive function `time` of the `Common LISP` language, we can also compare some program runtime indicators. Thus, for C_{26}^{13}, execution time for `Rosetta COMB` will be: `Run time:25.984375 sec` and for `Rosetta Comb Modified` – `Run time:10.859375 sec`. For C_{30}^{18}, these indicators will be: `Run time:261.70313 sec` and `Run time:105.4375 sec`, respectively.

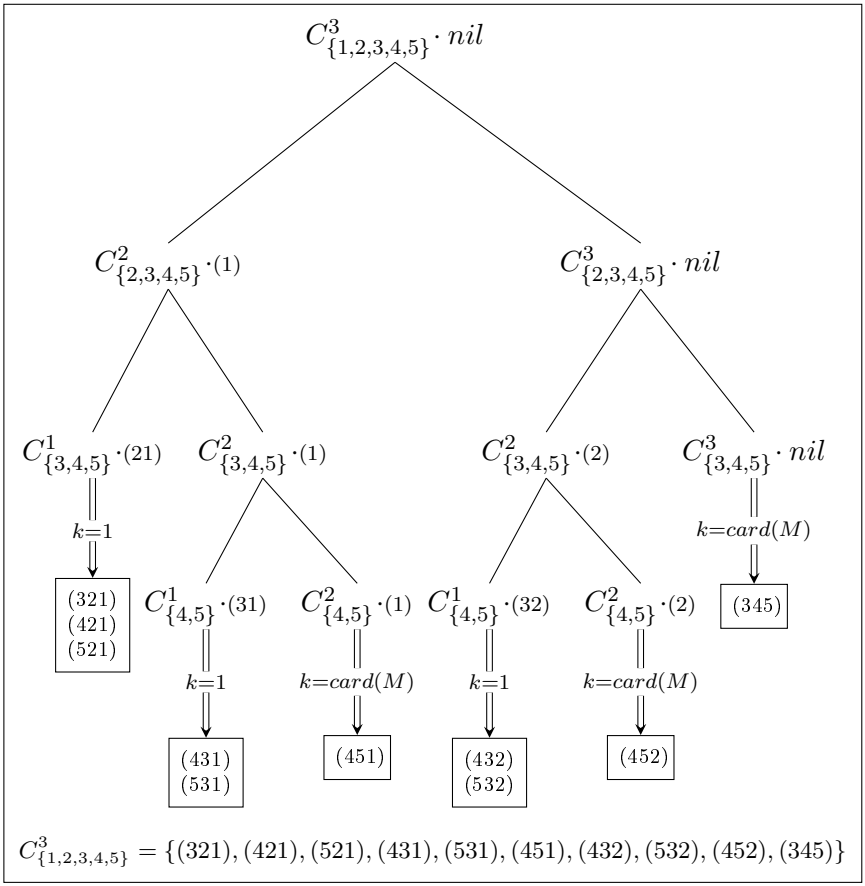

Figure 5: Schematic illustration of recursive calls for the `COMB modified` function, call C_5^3.

Some generated combinations will not appear in lexicographic order (descending in our case), but, we recall, the order of the elements in combination is irrelevant. If, however, in the modified function we substitute the call `(append l c)` with `(append(reverse l) c)`, we will get all combinations in lexicographic order.

In the Figure 5 we present the recursive calls scheme for the modified version `(comb 3 {1,2,3,4,5})`. This scheme helps us to better understand the proposed modified algorithm for combinations generation.

References

[1] Guy L. Steele, *Common Lisp the Language*, 2nd edition, Thinking Machines, Inc. Digital Press, ISBN: 1-55558-041-6, 1990, 1029 p.

[2] `http://sourceforge.net/projects/clisp/files/clisp/2.49/clisp-2.49-win32-mingw-big.exe/download`

[3] `http://rosettacode.org/wiki/Combinations#Common_Lisp`

Constantin Ciubotaru,

Received May 24, 2019
Accepted June 20, 2019

Constantin Ciubotaru
Vladimir Andrunachievici Institute of Mathematics and Computer Science
5, Academiei street, Chisinau, Republic of Moldova, MD 2028
Phone: (373 22) 73-80-73
E–mail: `chebotar@gmail.com`