Unranking algorithms applied to MuPAD

X. Molinero and J. Vives

***Abstract*—We present an improvement of the implementation of some unlabeled unranking algorithms of the open-source algebraic combinatorics package MuPAD-Combinat for the computer algebra system MuPAD. We compare our implementation with the current one. Moreover, we have also developed unranking algorithms applied to some unlabeled admissible operators that are not still implemented in the package MuPAD-Combinat. These algorithms are also able to develop some structures useful to generate molecules applied to chemistry and influence graphs applied to game theory and social networks, among other topics.**

***Index Terms*—Unranking Algorithms, MuPAD, Generating Molecules, Generating Influence Games.**

Unlabeled class	Specification
Binary trees	$\mathcal{B} = Z + \mathcal{B} \times \mathcal{B}$
Unary-binary trees or Motzkin trees	$\mathcal{M} = Z + Z \times \mathcal{M} + Z \times \mathcal{M} \times \mathcal{M}$
Integer partitions	$\mathcal{P} = \mathsf{Set}(\mathsf{Seq}(Z, \text{card} \geq 1))$
Integer compositions	$\mathcal{C} = \mathsf{Seq}(\mathsf{Set}(Z, \text{card} \geq 1))$
Non-ordered rooted trees or Rooted unlabeled trees	$\mathcal{T} = Z \times \mathsf{Set}(\mathcal{T})$
Binary sequences	$\mathcal{A} = \mathsf{Seq}(Z + Z)$
Non plane ternary trees	$\mathcal{D} = Z + \mathsf{Set}(\mathcal{D}, \text{card}{=}3)$
Integer partitions with distinct parts	$\mathcal{E} = \mathsf{PowerSet}(\mathsf{Seq}(Z, \text{card}{\geq}1))$

Fig. 1. Examples of unlabeled classes and their specifications

I. Introduction

The problem of *unranking* asks for the generation of the ith combinatorial object of size n in some combinatorial class $\mathcal{A}$, according to some well defined order among the objects of size n of the class. Efficient unranking algorithms have been devised for many different combinatorial classes, like binary and Cayley trees, Dyck paths, permutations, strings or integer partitions, but most of the work in this area concentrates in efficient algorithms for particular classes, whereas we aim at generic algorithms that apply to a broad family of combinatorial classes. The problem of unranking is intimately related with its converse, the *ranking* problem, as well as with the problems of random generation and exhaustive generation of all combinatorial objects of a given size. The interest of this whole subject is witnessed by the vast number of research papers and books that has appeared in over five decades (see, for instance, [24], [12], [9], [8], [11], [25], [10], [20], [19], [21], [3]).

[14], [13] designed *generic* unranking algorithms for a large family of combinatorial classes, namely, those which can be inductively built from the basic ϵ-class (a class which contains only one object of size 0), atomic classes (classes that contain only one object of size 1 or *atom*) and a collection of admissible combinatorial operators: disjoint unions, labeled and unlabeled products, sequence, set, etc. Now we use such techniques to implement those algorithms in MuPAD [2], [18]. In the open-source algebraic combinatorics package MuPAD-Combinat [1] for the computer algebra system MuPAD there are implemented the unranking for some admissible combinatorial operators, but now we improve such implementation for unlabeled unions and products (and sequences). Moreover, we have also implemented other operators as unlabeled sets and powersets (with and without restrictions).

The paper just considers unlabeled combinatorial classes and it is organized as follows. In Section II we briefly review basic definitions and concepts, the unranking algorithms and the theoretical analysis of their performance. Afterwards, from the computer algebra system MuPAD, we compare the required CPU time of our implementation with the required CPU time of the current implementation in the package MuPAD-Combinat. Moreover, we also explain our current and future work in this subject.

II. Preliminaries

As it will become apparent, all the unranking algorithms in this paper require an efficient algorithm for counting, that is, given a specification of a class and a size, they need to compute the number of objects with the given size. Hence, we will only deal with (some of) the so-called *admissible combinatorial classes* [6], [7]. Those are constructed from *admissible operators*, operations over classes that yield new classes, and such that the number of objects of a given size in the new class can be computed from the number of objects of that size or smaller sizes in the constituent classes. In this paper we just consider unlabeled objects (those whose atoms are indistinguishable[1]) built from these admissible combinatorial operators.

For unlabeled classes, the finite specifications are generated from the ϵ-class, atomic classes, and combinatorial operators including disjoint union ('+'), Cartesian product ('$\times$'), sequence ('Seq'), powerset ('PowerSet'), set ('Set')[2], and sequence, powerset and set (or multiset) with restricted cardinality. Figure 1 gives a few examples of unlabeled admissible classes.

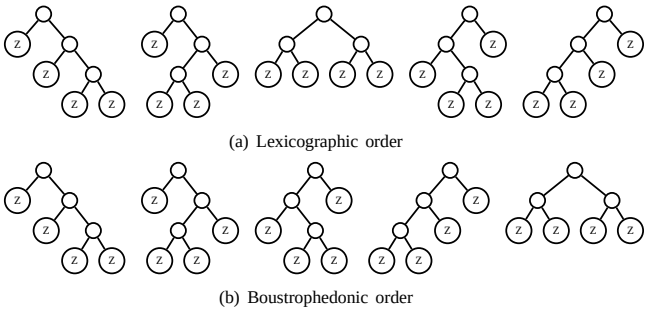
X. Molinero is with the Department of Applied Mathematics III, Universitat Politècnica de Catalunya, E-08240 Manresa, SPAIN. E-mail: xavier.molinero@upc.edu. X. Molinero was partially funded by grant MTM2012-34426/FEDER of the "Spanish Economy and Competitiveness Ministry".

J. Vives is with the Department of Design and Programming of Electronic Systems, Universitat Politècnica de Catalunya, E-08240 Manresa, SPAIN. E-mail: jvives@epsem.upc.edu.

[1] On the contrary, each of the n atoms of a *labeled* object of size n bears a distinct label drawn from the numbers 1 to n.

[2] Also denoted by 'multisets' (MultiSet) to emphasize that repetition is allowed.

ISBN: 978-1-61804-292-7



(a) Lexicographic order

(b) Boustrophedonic order

Fig. 2. Binary trees of size 4.

For the rest of this paper, we will use calligraphic uppercase letters to denote classes: $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, ... Given a class $\mathcal{A}$ and a size n, $\mathcal{A}_n$ will denote the subset of objects of size n in $\mathcal{A}$.

The order $\prec_{\mathcal{C}_n}$ among the objects of size n for a class $\mathcal{C} = \mathcal{A} + \mathcal{B}$ is naturally defined by $\gamma \prec_{\mathcal{C}_n} \gamma'$ if both γ and γ' belong to the same class (either $\mathcal{A}_n$ or $\mathcal{B}_n$) and $\gamma \prec \gamma'$ within their class, or if $\gamma \in \mathcal{A}_n$ and $\gamma' \in \mathcal{B}_n$. It is then clear that although $\mathcal{A} + \mathcal{B}$ and $\mathcal{B} + \mathcal{A}$ are isomorphic ("the same class"), these two specifications induce quite different orders. The unranking algorithm for disjoint unions compares the given rank with the cardinality of $\mathcal{A}_n$ to decide if the sought object belongs to $\mathcal{A}$ or to $\mathcal{B}$ and then solves the problem by recursively calling the unranking on whatever class ($\mathcal{A}$ or $\mathcal{B}$) is appropriate.

For Cartesian products the order in $\mathcal{C}_n = (\mathcal{A} \times \mathcal{B})_n$ depends on whether $\gamma = (\alpha, \beta)$ and $\gamma' = (\alpha', \beta')$ have first components of the same size. If $|\alpha| = |\alpha'| = j$ then we have $\gamma \prec_{\mathcal{C}_n} \gamma'$ if $\alpha \prec_{\mathcal{A}_j} \alpha'$ or $\alpha = \alpha'$ and $\beta \prec_{\mathcal{B}_{n-j}} \beta'$. But when $|\alpha| \neq |\alpha'|$, we must provide a criterion to order γ and γ'. The *lexicographic* order stems from the specification

$$\mathcal{C}_n = \mathcal{A}_0 \times \mathcal{B}_n + \mathcal{A}_1 \times \mathcal{B}_{n-1} + \ldots + \mathcal{A}_n \times \mathcal{B}_0,$$

in other words, the smaller object is that with smaller first component. On the other hand, the *boustrophedonic* order is induced by the specification

$$\mathcal{C}_n = \mathcal{A}_0 \times \mathcal{B}_n + \mathcal{A}_n \times \mathcal{B}_0 + \mathcal{A}_1 \times \mathcal{B}_{n-1} + \mathcal{A}_{n-1} \times \mathcal{B}_1 + \mathcal{A}_2 \times \mathcal{B}_{n-2} + \mathcal{A}_{n-2} \times \mathcal{B}_2 + \ldots,$$

in other words, we consider that the smaller pairs of total size n are those whose $\mathcal{A}$-component has size 0, then those with $\mathcal{A}$-component of size n, then those with $\mathcal{A}$-component of size 1, and so on. Figure II shows the lists of unlabeled binary trees of size 4 in lexicographic (a) and boustrophedonic order (b).

Of course, other orders are also possible, but they either do not help improving the performance of unranking or they are too complex to be useful or of general applicability.

For powersets, among some natural orders (see [14], [16]) we can choose

$$\mathsf{PowerSet}(\mathcal{A}) = \epsilon + \bigcup_{n>0} \bigcup_{j=1}^{n} \bigcup_{k=n\div j}^{1} \Big(\mathsf{PowerSet}(\mathcal{A}_j, \text{card}= k) \times \mathsf{PowerSet}_{n-kj}(\mathcal{A}_{>j})\Big)$$

where

$$\mathsf{PowerSet}(\mathcal{A}_j, \text{card}= k) = \bigcup_{\alpha \in \mathcal{A}_j} \Big(\alpha \times \mathsf{PowerSet}(\mathcal{A}_j^{(\succ \alpha)}, \text{card}= k-1)\Big),$$

being $\mathcal{A}^{(\succ \alpha)} = \{\alpha' \in \mathcal{A} : \alpha' \succ \alpha\}$, and $\mathsf{PowerSet}(\mathcal{A}_{>j})$ is a powerset with $\mathcal{A}$-components of size at least equal to $j+1$. Other orders described in [14], [16] do not change the complexity and they could also be easily adapted to our implementation.

For sets we have analogous isomorphisms but allowing repetitions.

The theoretical performance of these unranking algorithms is summarized in [16], [13].

Theorem 1: The worst-case time complexity of unranking for objects of size n in any admissible labeled class $\mathcal{A}$ using lexicographic ordering is of $\mathcal{O}(n^2)$ arithmetic operations.

Theorem 2: The worst-case time complexity of unranking for objects of size n in any admissible labeled class $\mathcal{A}$ using boustrophedonic ordering is of $\mathcal{O}(n \log n)$ arithmetic operations.

III. Our implementation v.s. MuPAD-Combinat implementation

In this section we compare our implementation[3] for unranking in MuPAD with the current implementation of the package MuPAD-Combinat(using MuPAD Pro 4.0). All our experiments run under Linux in a AMD64X2 4400 at 2.2 GHz with 4 Gb of RAM, and they use the basic facilities for counting already provided by the package MuPAD-Combinat.

For instance, the interface for binary trees has the following inputs:

```
spec := {B = Union(Z, BB), BB = Prod(B, B)};
p1 := combinat::
      decomposableObjects(spec, Lexi/Bous);
p1::unrank(rank, size);
```

where `spec` is the specification[4], `Lexi` or `Bous` forces the lexicographic or boustrophedonic order, respectively, and `rank` and `size` are the considered rank and size, respectively. Thus, the following commands provide all binary trees of size 8 in lexicographic order:

```
spec := {B = Union(Z, BB), BB = Prod(B, B)};
p1 := combinat::
      decomposableObjects(spec, Lexi);
for i from 0 to p1::count(8) - 1 do
  p1::unrank(i, 8);
end_for
```

[3]It is available on request from the first author; send an E-mail to xavier.molinero@upc.edu.

[4]The first class defined in the specification is the considered class (`B` in this case).

Lexicographic order				Boustrophedonic order			
Size	τ_T	τ'_T	ρ	Size	τ_T	τ'_T	ρ
25	1.24	4.55	0.27	25	0.90	2.95	0.30
50	3.63	11.95	0.30	50	2.50	6.52	0.38
75	6.08	20.17	0.30	75	4.27	10.24	0.41
100	9.27	29.65	0.31	100	6.20	13.97	0.44
125	12.46	39.98	0.31	125	6.99	17.91	0.39
150	16.54	51.73	0.31	150	8.39	20.80	0.40
175	21.37	66.16	0.32	175	9.67	24.90	0.38
200	26.85	79.93	0.33	200	11.63	28.87	0.40

TABLE I
AVERAGE CPU TIME (IN MILLISECONDS) FOR BINARY TREES, $\mathcal{B}$=Z+$\mathcal{B}\times\mathcal{B}$. THE CPU TIME REQUIRED TO CALCULATE THE PRE-COMPUTED TABLES IN OUR UNRANKING IS 380 MILLISECONDS.

Lexicographic order				Boustrophedonic order			
Size	τ_T	τ'_T	ρ	Size	τ_T	τ'_T	ρ
25	1.12	3.24	0.34	25	1.06	2.57	0.41
50	2.60	9.33	0.27	50	2.71	5.75	0.47
75	4.81	16.61	0.28	75	4.19	9.74	0.43
100	7.18	25.72	0.27	100	5.95	13.76	0.43
125	9.65	34.27	0.28	125	8.01	17.66	0.45
150	12.28	42.57	0.28	150	9.80	22.43	0.43
175	15.42	54.44	0.28	175	11.49	27.76	0.41
200	18.83	67.08	0.28	200	13.56	32.67	0.41

TABLE II
AVERAGE CPU TIME (IN MILLISECONDS) FOR MOTZKIN TREES, $\mathcal{M}$=Z+Z$\times\mathcal{M}$+Z$\times\mathcal{M}\times\mathcal{M}$. THE CPU TIME REQUIRED TO CALCULATE THE PRE-COMPUTED TABLES IN OUR UNRANKING IS 690 MILLISECONDS.

Notice that, in general, `p1::count(size)` returns the number of objects of `p1` with size `size`.

The selected collection for our experiments are two classical classes: binary trees ($\mathcal{B} = Z + \mathcal{B} \times \mathcal{B}$) and, unary-binary trees or Motzkin trees ($\mathcal{M} = Z + Z \times \mathcal{M} + Z \times \mathcal{M} \times \mathcal{M}$).

Essentially, we have used two techniques in our implementation. First, we have appropriately used the command `option remember`. Second, we have also used some pre–computed tables to store the counting of each considered class and size. The access to the indices of such tables is notably faster than the access to the command `count`. Tables I and II show the improvement of the average CPU time (in milliseconds). We have pre–computed the counting tables and, afterwards, we have generated 10000 random objects of the considered class and size. τ_T is our average time required to unrank a random rank of the considered class and size, τ'_T is MUPAD-COMBINAT average time required to unrank a random rank of the considered class and size, and ρ is the ratio τ_T/τ'_T. For any case, it looks as the improvements tend to be stable when the size n increase. For lexicographic binary trees it approaches to $\rho = 0.33$, for Boustrophedonic binary trees it approaches to $\rho = 0.40$, for lexicographic Motzkin trees it approaches to $\rho = 0.28$, and for Boustrophedonic Motzkin trees it approaches to $\rho = 0.41$. Thus, all results are satisfactorily better. Note that even the pre–computed tables require some CPU time, the average CPU time (when the number of generated objects increase) of our implementation substantially improves the previous one. We have meaningfully improved the average CPU time required to generate a random unranking: In lexicographic order, our implementation spends about 30% of the CPU time of the previous version; and, in boustrophedonic order, it spends about 40% of the CPU time of the previous version.

Sequences are done from unions and products, thus the timing improvements have similar advantages.

On the other hand, we have also done some experiments with classes that involve sets or cycles, for instance, we have considered the so-called *functional graphs* defined by $\mathcal{F}$= Set(Cycle($\mathcal{T}$)) with $\mathcal{T}$=$\times$(Z, Set($\mathcal{T}$)). In such cases, our implementation spends between 60% and 80% of the CPU time of the previous version.

A. Future implementation

The current implementation in MUPAD-COMBINAT does not consider all admissible combinatorial operators as well as restricted cardinalities in sets or powersets. We have added some of these operators in our implementation. In particular, we have considered admissible operators like

$$\varphi(\mathcal{B}, \operatorname{card} \tau\, k)$$

where $\varphi \in \{\mathsf{Seq}, \mathsf{Set}, \mathsf{PowerSet}\}$, $\tau \in \{\leq\ /\ =\geq\}$ and $k \in \mathbb{N}$.

By the way, the required average CPU time for the implemented operators is clearly competitive. Now one of the following open problems is to develop the corresponding pre–computed tables of counting for powersets and sets (see the described isomorphisms for powersets and sets in Section II).

IV. CONCLUSIONS AND FUTURE WORK

We have implemented in MUPAD the unranking applied to some basic unlabeled admissible combinatorial operators: disjoint unions, Cartesian products, and sequences. We are now working on the implementation for (unlabeled) powersets and sets (with and without restricted cardinalities).

Our implementation is making two main improvements for the unranking of unlabeled admissible classes in front of the implementation in the package MUPAD-COMBINAT. First, we have significantly reduced the average CPU time required to generate a random unranking. Second, we are programming more unlabeled admissible combinatorial operators (powersets and sets with and without restricted cardinalities).

Future work is to implement even more unlabeled admissible combinatorial operators (substitution, the open problem for unlabeled cycles, the union among non-disjoint classes, the intersection among classes, etc.).

Another line of research is to study similar operators but from the labeled point of view, that is, to consider that the nodes of the combinatorial structures of size n can be distinguished by labels from 1 to n.

The ranking, exhaustive and random generation should also be implemented [24], [12], [16].

On the other hand, these algorithms are also able to develop some structures useful to generate molecules [4], [5] applied to chemistry and influence graphs [17] applied to game theory and social networks, among other topics [22].

Finally, to what we know, it is still open to study the unranking, ranking and exhaustive generation of combinatorial structures from the viewpoint of genetic algorithms [15], [23]. Thus, it should be very interesting to establish some genetic algorithms to solve these problems.

ISBN: 978-1-61804-292-7

ACKNOWLEDGMENTS

We thank anonymous referees for their useful comments and suggestions that helped us to improve the contents of the paper.

REFERENCES

[1] MuPAD-Combinat – open-source algebraic combinatorics package for the computer algebra system MUPAD. URL: http://mupad-combinat.sourceforge.net/.

[2] C. Creutzig and W. Oevel. MUPAD *Tutorial*. SciFace Software (SciFace), Paderborn, 2004.

[3] S. Even. *Combinatorial Algorithms*. MacMillan, New York, 1973.

[4] P. Flajolet and B. Salvy. Computer algebra libraries for combinatorial structures. *J. Symbolic Computation*, 20:653–671, 1995.

[5] P. Flajolet, B. Salvy, and P. Zimmermann. Lambda-upsilon-omega: The 1989 cookbook. Technical Report 1073, INRIA, 1989.

[6] P. Flajolet and R. Sedgewick. The average case analysis of algorithms: Counting and generating functions. Technical Report 1888, INRIA, 1993.

[7] P. Flajolet and J.S. Vitter. Average-case Analysis of Algorithms and Data Structures. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 9. North-Holland, 1990.

[8] D.L. Kreher and D.R. Stinson. *Combinatorial Algorithms: Generation, Enumeration and Search*. CRC Press LLC, 1999.

[9] Greg Kuperberg, Shachar Lovett, and Ron Peled. Probabilistic existence of regular combinatorial structures. *CoRR*, abs/1302.4295, 2013.

[10] J. Liebehenschel. Ranking and unranking of lexicographically ordered words: An average-case analysis. *J. of Automata, Languages and Combinatorics*, 2(4):227–268, 1997.

[11] J. Liebehenschel. Ranking and unranking of a generalized dyck language and the application to the generation of random trees. In *The Fifth International Seminar on the Mathematical Analysis of Algorithms*, Bellaterra (Spain), 1999.

[12] A. Lorenz and Y. Ponty. Non-redundant random generation algorithms for weighted context-free languages. *Theoretical Computer Science, Elsevier, 2013, Generation of Combinatorial Structures*, 502:177–194, 2013.

[13] C. Martínez and X. Molinero. A generic approach for the unranking of labeled combinatorial classes. *Random Structures & Algorithms*, 19(3-4):472–497, 2001.

[14] C. Martínez and X. Molinero. Efficient iteration in admissible combinatorial classes. *Theoretical Computer Science*, 346(2–3):388–417, November 2005.

[15] M. Mitchell. *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. The MIT Press.

[16] X. Molinero. *Ordered Generation of Classes of Combinatorial Structures*. PhD thesis, Universitat Politècnica de Catalunya, November 2005.

[17] X. Molinero, F. Riquelme, and M. J. Serna. Cooperation through social influence. *European Journal of Operation Research*, 242(3):960–974, May 2015.

[18] MUPAD: The computer algebra system. URL: http://es.mathworks.com/discovery/mupad.html.

[19] A. Nijenhuis and H.S. Wilf. *Combinatorial Algorithms: For Computers and Calculators*. Academic Press, Inc., 1978.

[20] J.M. Pallo. Enumerating, ranking and unranking binary trees. *The Computer Journal*, 29(2):171–175, 1986.

[21] E.M. Reingold, J. Nievergelt, and N.Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[22] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley, Reading, MA, 1996.

[23] R. Keller W. Banzhaf, P. Nordin and F. Francone. *Genetic Programming An Introduction*. San Francisco, CA: Morgan Kaufmann, 1998.

[24] Y. Wei. The grouping combinaton generating algorithm. In *Proceedings of the International Conference on Computer, Network Security and Communication Engineering (CNSCE 2014)*, pages 670–674, 2014.

[25] H.S. Wilf. East side, west side ... an introduction to combinatorial families- with MAPLE programming. Technical report, 1999. URL: http://www.cis.upenn.edu/~wilf/lecnotes.html.