Space Complexity of Enumeration

Yann Strozecki[1]

[1]Université Paris Saclay, UVSQ, DAVID laboratory, France

I. Introduction

Enumeration algorithms solve the task of listing a set of elements and are used to count objects, to solve optimization problems or to build a collection of objects to be interpreted by another algorithm or a human expert. The dynamic nature of enumeration algorithms is taken into account when characterizing their complexity: we consider the *incremental time*, that is the time to generate the first k solutions and the *delay* between two consecutive solutions. When a problem can be solved by an algorithm that produces the first k solutions from an input of size n, in time $k^a p(n)$ with p a polynomial, the problem is in polynomial incremental time and belongs to the class IncP_a. If the algorithm has a polynomial delay, then it is in the class DelayP (for more details on complexity classes for enumeration see [1], [2]).

A problem is considered tractable if it belongs to one of the classes we have introduced. Indeed, bounding the delay or the incremental time implies a bound on the total time to generate all solutions function of the number of solutions. Moreover, it guarantees that a subset of all solutions can be generated efficiently if we do not want them all.

In practice, bounding the space used by enumeration algorithms seems as important as bounding their delay as it is already noted in [3]. Therefore, much effort has been devoted ensuring that polynomial delay methods run in polynomial space [4], [5], [6], [7], [8]. We denote by DelayP^{poly} (respectively IncP_a^{poly}) the problems solved by polynomial delay (resp. by polynomial incremental time) and polynomial space algorithms.

It is possible to turn an algorithm in incremental linear time into an algorithm in polynomial delay, by delaying the output of solutions using a buffer. However, this method requires exponential space for the buffer, which makes it impractical. Recently, we have provided an algorithm, called geometric amortization, which turns any incremental linear time algorithm into a polynomial delay algorithm with a polynomial space overhead: $\mathrm{IncP}_1^{poly} = \mathrm{DelayP}^{poly}$ [9]. We then tried to apply this result to the many problems of the literature using a buffer to amortize the delay, hoping to show that many of them only require polynomial space with our method. However, in most examples, the buffer is also used to eliminate duplicates and it seems inevitable. Hence, in this extended abstract, we try to understand better the cost of duplicates elimination by collecting and adapting several folklore results. We also give a few new structural results on space complexity in enumeration.

II. Systematic Elimination of Duplicates

Dealing with duplicated solutions is a major problem in enumeration. One possible source of duplicates is when the problem can be interpreted as a non-disjoint union of sets of solutions, where each set can be enumerated efficiently. In this case, an efficient method to obtain an algorithm without duplicates and no space overhead is described in [10]. Often, we know how to enumerate a set of elements efficiently, but we only want to output equivalence classes of this set, for instance, graphs up to isomorphism [11] or solutions of simple dynamic programs [12]. We say that an algorithm solves an enumeration problem Π_A with repetitions if it outputs all solutions at least once, whereas an algorithm solving Π_A outputs all solutions exactly once. We say that Π_A is in IncP_a with repetitions, if there is an algorithm and a polynomial p, such that the algorithm outputs at least k distinct solutions in time $k^a p(n)$.

Theorem 1. *Let Π_A be a problem in IncP_a with repetitions then $\Pi_A \in \mathrm{IncP}_a$ and exponential space.*

Proof. We simulate the algorithm solving Π_A with incremental time $k^a p(n)$. The algorithm maintain a trie of all output solutions. Each time the simulation produce a solution, we test whether it is in the trie and if not the solution is output and added to the trie. The overhead is only proportional to the size of a solution, which is polynomial in the input, hence the described algorithm is in IncP_a. □

A. Forward-search

We can avoid exponential space when eliminating duplicates in an algorithm with repetitions, but it increases the incremental time significantly.

Theorem 2. *Let Π_A be a problem in IncP_a^{poly} with repetitions then $\Pi_A \in \mathrm{IncP}_{2a}^{poly}$. If the number of occurrences of each solution is polynomially bounded, then $\Pi_A \in \mathrm{IncP}_{a+1}^{poly}$.*

Proof. Simulate the algorithm solving Π_A with incremental time $k^a p(n)$. Each time t such that a solution y is produced, we run a new simulation up to time $t-1$ and output y only if y is not output in this second simulation. This algorithm produces at least k distinct solutions in time $O(k^{2a} p(n)^2)$.

When each solution is repeated at most $q(n)$ times, we can bound the number of times the previous algorithm must simulate the enumeration from the start. In time $O(q(n)p(n)k^{a+1})$, k solutions are produced by the simulation. □

Sometimes you may simulate the algorithm for Π_A backward from any point of time, for instance when the algorithm is a tree traversal of solutions or partial solutions. In the previous method, we may simulate the algorithm backward to check whether the solution has already been output. Hence, all identical solutions produced up to time t require together a time $O(t)$ to be checked. Therefore, the algorithm without repetition is of complexity $O(p(n)k^{a+1})$.

B. Time/Space Trade-off

We can obtain a tradeoff between time and space when eliminating duplicates, however we can only bound the total time to generate all solutions and not the incremental time.

Theorem 3. *Let $\lambda(n)$ be any function and let Π_A be a problem of total time $t(n)$ using a space $s(n)$ and each solution is of size at most $p(n)$. Then there is an algorithm in total time $O(t(n)p(n) * \lceil t(n)/\lambda(n) \rceil)$ and space $s(n) + \lambda(n)p(n)$.*

Proof. Simulate the algorithm for Π_A and store the first $\lambda(n)$ solutions in lexicographic order, using for instance a trie. The size of a solution is bounded by $p(n)$ for some polynomial p. Simulate the algorithm back again to obtain the next $\lambda(n)$ solutions and so on until all solutions are produced. □

This method has been proposed for turning a uniform sampler into a probabilistic enumeration algorithm in the Thesis of Leslie Ann Goldberg [3]. She also proves that any algorithm using a black box uniform sampler to enumerate all solutions are such that the product of their delay and their space is lower bounded by the number of solutions to output up to a polynomial factor. A similar theorem can be proved for the elimination of duplicates from a black box enumeration algorithm with repetitions, proving that the space use is inevitable.

C. Tractable Elimination of Duplicates

There are a few settings, in which duplicates can be eliminated efficiently without space overhead or a large time overhead. Typically, algorithms list duplicates because they generate many distinct objects all in the same equivalence class and the objective is to generate all equivalence classes.

To obtain an efficent algorithm generating equivalence classes from an algorithm generating all objects, we need to be able to find a canonical representative in polynomial time and that the classes are of polynomial size. Then, we generate all objects and filter them using a canonicity test, which gives an algorithm with a bounded total time. If we further want a bound on the incremental time or the delay, we must be able to tell which of the objects in an equivalence class is generated first.

We can give three examples where such methods could be used:

- The elements generated are of the form $(y_1, \ldots, y_k)$ for a constant k but we only want the sets $\{y_1, \ldots, y_k\}$ (without the order).
- The elements generated are of the form $(y_1, \ldots, y_k, y_{k+1})$, such that y_{k+1} is of logarithmic size but we only want the elements $(y_1, \ldots, y_k, y_k)$ (the last element is projected away).
- The average size of an equivalence class is small and we can list elements in small equivalence classes first (see e.g. the generation of graphs up to isomorphism [3]).

III. Structural Results on Space Complexity

We try to understand the power of using more memory in general.

A. Is Exponential Space Useful ?

Let Enum^{poly} be the set of all enumeration problems solvable using a polynomial space machine, an equivalent of PSPACE for enumeration. The class EnumP (see [1]) is the set of enumeration problems whose solutions are of polynomial size and can be checked in polynomial time. What can we say of the relative power of these classes?

Theorem 4. 1) $\mathrm{EnumP} \subseteq \mathrm{Enum}^{poly}$
2) $\mathsf{P} = \mathsf{PSPACE}$ *if and only if* $\mathrm{EnumP} = \mathrm{Enum}^{poly}$

Proof. 1) Let $\Pi_A \in \mathrm{EnumP}$, by definition we can check whether an element y is a solution of Π_A for some instance x in polynomial time and thus in polynomial space. Moreover, the solutions are of polynomial size, hence there are at most an exponential number of potential solutions, which can easily be generated in polynomial space. Therefore, the algorithm which lists all potential solutions and checks them before outputting them is in polynomial space.

2) Consider any PSPACE problem A, from it we define the enumeration problem Π_A whose set of solutions is $\{1\}$ if the instance x is in A and $\{0\}$ otherwise. If $\mathrm{EnumP} = \mathrm{Enum}^{poly}$, then given an instance x, we can test in polynomial time whether 1 is a solution proving that $A \in \mathsf{P}$.

Consider now a problem $\Pi_A \in \mathrm{Enum}^{poly}$. We consider the decision problem A: given (x, y) is y a solution of the problem Π_A on the instance x. Using the algorithm enumerating all solutions of Π_A in polynomial space, we get an algorithm to solve A in polynomial space. We assume $\mathsf{P} = \mathsf{PSPACE}$, hence $A \in \mathsf{P}$, which implies that we can test whether an element is a solution in polynomial time. Since Π_A can be solved in polynomial space, the solutions output must themselves be of polynomial size in the input, hence we have $\Pi_A \in \mathrm{EnumP}$. □

The same theorem can be proved for the polynomial hierarchy defined in [13].

B. Exponential Space and Tractable Problem

A more practical question is to understand whether exponential space enlarges the classes IncP_i, which correspond to tractable problems. We relax the constraint that solutions can be checked in polynomial time and we denote the corresponding classes by $\tilde{\mathrm{IncP}}_i$. Remark that it does not seem possible to characterize the class IncP^{poly} by a search problem asking to find a solution not in a given set (see [2]), which seems to make it different from IncP.

Theorem 5. *If* $\mathsf{EXP} \neq \mathsf{PSPACE}$ *then* $\tilde{\mathrm{IncP}}_1 \not\subset \mathrm{Enum}^{poly}$.

Proof. Let A be a problem in EXP, which can be solved in time $O(2^{p(n)})$, with p a polynomial. Let Π_A be the enumeration problem which on the instance x has solutions $\{1, 2, \ldots, 2^{p(|x|)}\}$ and also the special solution $\sharp$ if $x \in A$. This problem is in $\tilde{\mathrm{IncP}}_1$, since we can enumerate the trivial solutions $\{1, 2, \ldots, 2^{p(|x|)}\}$ in time $O(2^{p(n)})$ with constant delay and then in time $O(2^{p(n)})$ we can decide whether $x \in A$ and then output $\sharp$ or not.

If we assume that $\tilde{\mathrm{IncP}}_1 \subset \mathrm{Enum}^{poly}$, then we can solve Π_A in polynomial space. The enumeration algorithm can be used to solve the problem A in polynomial space, by detecting whether $\sharp$ is output. Hence $A \in \mathsf{PSPACE}$. □

Open Problem 1. *Can we relax the hypothesis of the previous problem to prove equivalence with a classical complexity hypothesis?*

Open Problem 2. *Can we prove* $\mathrm{IncP}_i \neq \mathrm{IncP}_i^{poly}$*, assuming some complexity hypothesis?*

More than understanding the separation of complexity classes, we would like to understand when using exponential space strictly decreases the incremental time needed to solve specific problems.

Open Problem 3. *The enumeration of the circuits of a binary matroid is in* IncP_2*, using a saturation algorithm [14], which intrinsically relies on exponential memory. Can we prove that the problem is in* IncP_2^{poly} *or even in* IncP^{poly}*?*

Acknowledgments: The author thanks Florent Capelli and Arnaud Mary for many discussions on the complexity of enumeration.

References

[1] Y. Strozecki, "Enumeration complexity," *Bulletin of EATCS*, vol. 1, no. 129, 2019.

[2] Y. Strozecki, *Enumeration Complexity: Incremental Time, Delay and Space*. Habilitation thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2021.

[3] L. A. Goldberg, *Efficient algorithms for listing combinatorial structures*. PhD thesis, University of Edinburgh, UK, 1991.

[4] E. L. Lawler, J. K. Lenstra, and A. Rinnooy Kan, "Generating all maximal independent sets: NP-hardness and polynomial-time algorithms," *SIAM Journal on Computing*, vol. 9, no. 3, pp. 558–565, 1980.

[5] D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Applied Mathematics*, vol. 65, no. 1-3, pp. 21–46, 1996.

[6] S. Cohen, B. Kimelfeld, and Y. Sagiv, "Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties," *Journal of Computer and System Sciences*, vol. 74, no. 7, pp. 1147–1159, 2008.

[7] A. Conte and T. Uno, "New polynomial delay bounds for maximal subgraph enumeration by proximity search," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1179–1190, 2019.

[8] C. Brosse, V. Limouzy, and A. Mary, "Polynomial delay algorithm for minimal chordal completions," in *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France* (M. Bojanczyk, E. Merelli, and D. P. Woodruff, eds.), vol. 229 of *LIPIcs*, pp. 33:1–33:16, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[9] F. Capelli and Y. Strozecki, "Geometric amortization of enumeration algorithms," *arXiv preprint arXiv:2108.10208*, 2021.

[10] A. Durand and Y. Strozecki, "Enumeration complexity of logical query problems with second-order variables," in *CSL*, pp. 189–202, 2011.

[11] L. A. Goldberg, "Efficient algorithms for listing unlabeled graphs," *J. Algorithms*, vol. 13, no. 1, pp. 128–143, 1992.

[12] Y. Wang, A. Mary, M. Sagot, and B. Sinaimeri, "A general framework for enumerating equivalence classes of solutions," in *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)* (P. Mutzel, R. Pagh, and G. Herman, eds.), vol. 204 of *LIPIcs*, pp. 80:1–80:14, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[13] N. Creignou, M. Kröll, R. Pichler, S. Skritek, and H. Vollmer, "A complexity theory for hard enumeration problems," *Discrete Applied Mathematics*, vol. 268, pp. 191–209, 2019.

[14] L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, and K. Makino, "On the complexity of some enumeration problems for matroids," *SIAM Journal on Discrete Mathematics*, vol. 19, no. 4, pp. 966–984, 2005.