Space in Enumeration

Yann Strozecki

WEPA 2022

The space question

Observation: Space use may be a bottleneck in practical enumeration algorithms.

The space question

Observation: Space use may be a bottleneck in practical enumeration algorithms.

How can we reduce space use in enumeration?

What to do?

Not in this talk:

- Design specific algorithms, typically going from supergraph method to reverse search.
- Find alternative to enumeration to not generate all solutions.

What to do?

Not in this talk:

- Design specific algorithms, typically going from supergraph method to reverse search.
- Find alternative to enumeration to not generate all solutions.

In this talk:

- How space change the complexity landscape.
- Dealing with duplicates with small space overhead.

Framework

An enumeration problem A is a function which associates to each input a set of solutions $A(x)$.

An enumeration algorithm must generate every element of $A(x)$ one after the other without repetition.

Computation model: RAM with uniform cost measure and an OUPTPUT instruction. Support efficient data structures.

Complexity measures:

- total time
- incremental time
- delay
- space

Parameters:

- input size
- output size
- single solution size

Two Classical Complexity Classes

Equivalent of NP for enumeration:

Definition

EnumP is the set of enumeration problems whose solutions are of polynomial size and can be checked in polynomial time.

Enum·SAT is EnumP-complete.

Two Classical Complexity Classes

Equivalent of NP **for enumeration:**

Definition

$\textsc{EnumP}$ is the set of enumeration problems whose solutions are of polynomial size and can be checked in polynomial time.

$\textsc{Enum}\cdot SAT$ is $\textsc{EnumP}$-complete.

Equivalent of PSPACE **for enumeration:**

Definition

$\textsc{Enum}^{poly}$ is the set of all enumeration problems solvable using a polynomial space machine

$\textsc{Enum}\cdot QSAT$ is $\textsc{Enum}^{poly}$-complete.

Relevance of Polynomial Space in Application of Enumeration

Number of solutions can be *superpolynomial* in the input size $\Rightarrow$ generation in polynomial space of the exact and explicit solution set is not relevant.

Relevance of Polynomial Space in Application of Enumeration

Number of solutions can be *superpolynomial* in the input size $\Rightarrow$ generation in polynomial space of the exact and explicit solution set is not relevant.

Polynomial space is relevant when we compute something from the set of solutions:

- maximum (optimization)
- counting
- statistics
- intermediary objects to generate another set

Is Exponential Space Useful?

Theorem

1. $\textsc{EnumP} \subseteq \textsc{Enum}^{poly}$
2. P $\neq$ PSPACE *if and only if* $\textsc{EnumP} \neq \textsc{Enum}^{poly}$

Proof:

1) Solutions of $\textsc{EnumP}$ problems are of polynomial size and can be checked in polynomial time.

2) Encode a PSPACE problem as the enumeration of its truth value. Conversely a polynomial space enumeration algorithm implies a polynomial space verification algorithm.

Transfer from Classical Complexity

Adaptation from decision complexity:

- Hierarchy theorem in space
- Non-deterministic polynomial space can be defined in several ways
- Savitch like theorem: Non-deterministic polynomial space equal to $\textsc{Enum}^{poly}$
- Logarithmic space enumeration can be defined
- An Immerman-Szelepcsényi like theorem for non deterministic logarithmic space

Incremental time

A machine solves A in incremental time $f(t, n)$ if, on every input x of size n and $t \leq |A(x)|$, it enumerates t elements of $A(x)$ in time $f(t, n)$.

Definition (Incremental polynomial time hierarchy)

A problem A is in $\textsc{IncP}_a$ if there is a machine M which solves it in incremental time $O(t^a n^b)$ for some constant b. When M is in polynomial space, then $A \in \textsc{IncP}_a^{poly}$.

- Minimal tractability: Polynomial incremental time.
- Real tractability: linear incremental time (polynomial delay) and polynomial space, the class $\textsc{IncP}_1^{poly}$.

Polynomial Space and Tractable Classes

Not possible to systematically get rid of exponential space in efficient algorithms.

Theorem

If EXP $\neq$ PSPACE *then* $\textsc{IncP}_1 \not\subset \textsc{Enum}^{poly}$.

Proof:

From $A \in$ EXP build an enumeration problem $\textsc{Enum}{\cdot}B$. It has a solution witnessing the answer to A and an exponential number of trivial solutions, hence $\textsc{Enum}{\cdot}B \in \textsc{IncP}_1$. If $\textsc{Enum}{\cdot}B \in \textsc{Enum}^{poly}$, we get that $A \in$ PSPACE.

Polynomial Space and Tractable Classes

Not possible to systematically get rid of exponential space in efficient algorithms.

Theorem

If EXP $\neq$ PSPACE *then* $\textsc{IncP}_1 \not\subset \textsc{Enum}^{poly}$.

Proof:

From $A \in$ EXP build an enumeration problem $\textsc{Enum}{\cdot}B$. It has a solution witnessing the answer to A and an exponential number of trivial solutions, hence $\textsc{Enum}{\cdot}B \in \textsc{IncP}_1$. If $\textsc{Enum}{\cdot}B \in \textsc{Enum}^{poly}$, we get that $A \in$ PSPACE.

Open Question: Can we get the same theorem for checkable version of $\textsc{IncP}_1$?

Open Question: Find a real problem for which we can prove an $\textsc{IncP}_a^{poly}$ lower bound.

Regularization

In one case, we can "remove" exponential space: regularization.

Theorem (Capelli, S.)

$\textsc{IncP}_1^{poly} = \textsc{DelayP}^{poly}$

Many problems become polynomial delay and polynomial space with a small overhead?

Regularization

In one case, we can "remove" exponential space: regularization.

Theorem (Capelli, S.)

$\textsc{IncP}_1^{poly} = \textsc{DelayP}^{poly}$

Many problems become polynomial delay and polynomial space with a small overhead?

No: problems using a datastructure for regularization also use it for duplicates elimination!

Regularization

In one case, we can "remove" exponential space: regularization.

Theorem (Capelli, S.)

$\textsc{IncP}_1^{poly} = \textsc{DelayP}^{poly}$

Many problems become polynomial delay and polynomial space with a small overhead?

No: problems using a datastructure for regularization also use it for duplicates elimination!

Open Question: Examples of regularization and duplicates elimination in exponential space in the litterature?

Eliminating duplicates

Duplicates do not matter when solving an *optimization problem* using enumeration.

You should eliminate duplicates when:

- Computing statistics on the solution set.
- Generating a temporary set of solutions to be used by something else.
- Generating a few examples solutions $\rightarrow$ easy.

Algorithm with duplicates

An algorithm solves an enumeration problem $\textsc{Enum}\cdot A$ with duplicates if it outputs all solutions at least once.

Definition

$\textsc{Enum}\cdot A$ is in $\textsc{IncP}_a$ with duplicates, if there is an algorithm and a polynomial p, such that the algorithm outputs at least t distinct solutions in time $t^a p(n)$.

Algorithm with duplicates

An algorithm solves an enumeration problem $\textsc{Enum}\cdot A$ with duplicates if it outputs all solutions at least once.

Definition

$\textsc{Enum}\cdot A$ is in $\textsc{IncP}_a$ with duplicates, if there is an algorithm and a polynomial p, such that the algorithm outputs at least t distinct solutions in time $t^a p(n)$.

Theorem

Let $\textsc{Enum}\cdot A$ *be a problem in* $\textsc{IncP}_a$ *with repetitions then* $\textsc{Enum}\cdot A \in \textsc{IncP}_a$ *and exponential space.*

Proof: Use a trie, overhead in the size of a single solution.

Eliminating Duplicates Without Space

Theorem

Let $\textsc{Enum}\cdot A$ *be a problem in* $\textsc{IncP}_a^{poly}$ *with duplicates then* $\textsc{Enum}\cdot A \in \textsc{IncP}_{2a}^{poly}$.

Proof:

Simulate the algorithm solving $\textsc{Enum}\cdot A$ 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 $O(k^{2a}p(n)^2)$.

Eliminating Duplicates Faster (Forward)

Theorem

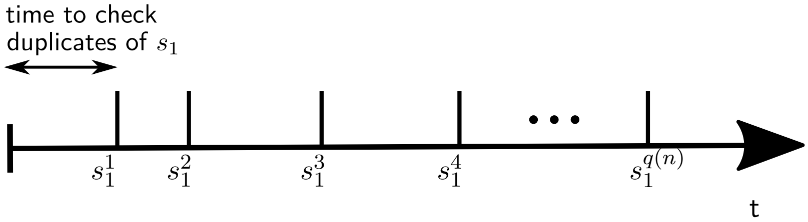
Let $\textsc{Enum}{\cdot}A$ *be a problem in* $\textsc{IncP}_a^{poly}$ *with polynomial number of duplicates then* $\textsc{Enum}{\cdot}A \in \textsc{IncP}_{a+1}^{poly}$.

Eliminating Duplicates Faster (Forward)

Theorem

Let $\textsc{Enum}{\cdot}A$ *be a problem in* $\textsc{IncP}_a^{poly}$ *with polynomial number of duplicates then* $\textsc{Enum}{\cdot}A \in \textsc{IncP}_{a+1}^{poly}$.

Same algorithm, at most $q(n)$ occurences of each solution. From incremental time $O(p(n)k^a)$ with duplicate to in incremental time $O(q(n)p(n)k^{a+1})$.



Eliminating Duplicates Faster (Backward)

Theorem

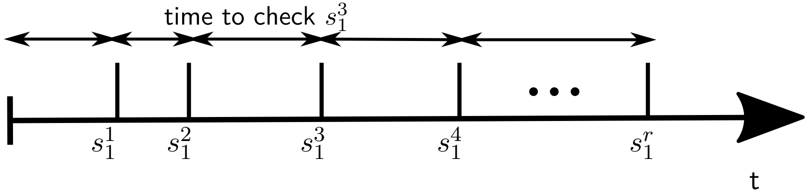
Let $\textsc{Enum}\cdot A$ be a problem in $\textsc{IncP}_a^{poly}$ with an algorithm which can be computed backwards then $\textsc{Enum}\cdot A \in \textsc{IncP}_{a+1}^{poly}$.

Eliminating Duplicates Faster (Backward)

Theorem

Let $\textsc{Enum}{\cdot}A$ *be a problem in* $\textsc{IncP}_a^{poly}$ *with an algorithm which can be computed backwards then* $\textsc{Enum}{\cdot}A \in \textsc{IncP}_{a+1}^{poly}$.

For each solution output, check whether it appears before in the computation, by doing it backwards from this point in time. From incremental time $O(p(n)k^a)$ with duplicates to incremental time $O(p(n)k^{a+1})$.



Space-Time Trade-off to Eliminate Duplicates

From a result of Leslie Ann Goldberg on turning a random generator to an enumeration. Bound on the total time.

Theorem

Let $\lambda(n)$ be any function and let $\textsc{Enum}{\cdot}A$ be a problem which can be solved with duplicates in total time $t(n)$ using a space $s(n)$ and each solution 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 $s(n) + \lambda(n)p(n)$.

Space-Time Trade-off to Eliminate Duplicates

From a result of Leslie Ann Goldberg on turning a random generator to an enumeration. Bound on the total time.

Theorem

Let $\lambda(n)$ be any function and let $\textsc{Enum}{\cdot}A$ be a problem which can be solved with duplicates in total time $t(n)$ using a space $s(n)$ and each solution 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 $s(n) + \lambda(n)p(n)$.

Probabilistic settings: product of space and delay larger than number of solutions.

Space-Time Trade-off to Eliminate Duplicates

From a result of Leslie Ann Goldberg on turning a random generator to an enumeration. Bound on the total time.

Theorem

Let $\lambda(n)$ be any function and let $\textsc{Enum}{\cdot}A$ be a problem which can be solved with duplicates in total time $t(n)$ using a space $s(n)$ and each solution 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 $s(n) + \lambda(n)p(n)$.

Probabilistic settings: product of space and delay larger than number of solutions.

Open Question: prove the lower bound in the deterministic, black box settings. Adapt it to incremental time.

Efficient methods to remove duplicates

Removing duplicates in $\textsc{IncP}_1^{poly}$, in *special cases*:

1. Solutions are a polynomial union of sets that can be generated in $\textsc{IncP}_1^{poly}$.
2. Solutions are a quotient by an equivalence relation. Equivalence classes are polysize and polytime canonicity test.
   - $(y_1, \dots, y_k) \rightarrow \{y_1, \dots, y_k\}$
   - $(y_1, \dots, y_k) \rightarrow (y_1, \dots, y_{k-1})$
3. Equivalence class are polysize *on average* and we can list small equivalence classes first.

Efficient methods to remove duplicates

Removing duplicates in $\textsc{IncP}_1^{poly}$, in *special cases*:

1. Solutions are a polynomial union of sets that can be generated in $\textsc{IncP}_1^{poly}$.
2. Solutions are a quotient by an equivalence relation. Equivalence classes are polysize and polytime canonicity test.
   - $(y_1, \ldots, y_k) \to \{y_1, \ldots, y_k\}$
   - $(y_1, \ldots, y_k) \to (y_1, \ldots, y_{k-1})$
3. Equivalence class are polysize *on average* and we can list small equivalence classes first.

Open Question: Another method to eliminate duplicates without space?

Thanks.