Approximations Algorithms

(for Database Researchers)

Masses de données distribuées, 9 June 2016

Licence de droits d'usage

Pierre Senellart

TELECOM ParisTech

Optimization and computation problems

Let f be a function $f : \mathcal{X} \to \mathbb{R}$ (often, but not always, a function that counts something). We focus in this talk on:

Optimization and computation problems

Let f be a function $f : \mathcal{X} \to \mathbb{R}$ (often, but not always, a function that counts something). We focus in this talk on:

Optimization problems. Given a set of objects S, find some subset $X \subseteq S$ such that $f(X)$ is minimal (or maximal) among all X satisfying some conditions.

Optimization and computation problems

Let f be a function $f : \mathcal{X} \to \mathbb{R}$ (often, but not always, a function that counts something). We focus in this talk on:

Optimization problems. Given a set of objects S, find some subset $X \subseteq S$ such that $f(X)$ is minimal (or maximal) among all X satisfying some conditions.

Given a database D, find some set of tuples X such that $f(X)$ is minimal among all X satisfying some conditions.

Optimization and computation problems

Let f be a function $f : \mathcal{X} \to \mathbb{R}$ (often, but not always, a function that counts something). We focus in this talk on:

Optimization problems.
: Given a set of objects S, find some subset $X \subseteq S$ such that $f(X)$ is minimal (or maximal) among all X satisfying some conditions.
Given a database D, find some set of tuples X such that $f(X)$ is minimal among all X satisfying some conditions.

Computation problems.
: Given a set of objects S, compute the value of $f(S)$.

Licence de droits d'usage

Optimization and computation problems

Let f be a function $f : \mathcal{X} \to \mathbb{R}$ (often, but not always, a function that counts something). We focus in this talk on:

Optimization problems.

Given a set of objects S, find some subset $X \subseteq S$ such that $f(X)$ is minimal (or maximal) among all X satisfying some conditions.

Given a database D, find some set of tuples X such that $f(X)$ is minimal among all X satisfying some conditions.

Computation problems.

Given a set of objects S, compute the value of $f(S)$.

Given a database D, compute the value of $f(D)$.

Licence de droits d'usage

Examples of optimization problems

Examples

Maximum Matching. Given a set of tasks and a set of workers with preferences of workers on tasks, find an assignment for all tasks that maximizes satisfaction.

Licence de droits d'usage

Examples of optimization problems

Examples

Maximum Matching. Given a set of tasks and a set of workers with preferences of workers on tasks, find an assignment for all tasks that maximizes satisfaction.

Set Cover. Given a set of people, each with fluency in various languages, find a group of people of minimum size who can speak all the languages.

Examples of optimization problems

Examples

Maximum Matching. Given a set of tasks and a set of workers with preferences of workers on tasks, find an assignment for all tasks that maximizes satisfaction.

Set Cover. Given a set of people, each with fluency in various languages, find a group of people of minimum size who can speak all the languages.

Vertex Cover. In a city, find a set of minimum size of road intersections where to put street cameras such that all roads are covered.

Examples of optimization problems

Examples

Maximum Matching. Given a set of tasks and a set of workers with preferences of workers on tasks, find an assignment for all tasks that maximizes satisfaction.

Set Cover. Given a set of people, each with fluency in various languages, find a group of people of minimum size who can speak all the languages.

Vertex Cover. In a city, find a set of minimum size of road intersections where to put street cameras such that all roads are covered.

Inconsistent Data Repair. Given a database inconsistent w.r.t. fixed integrity constraints, find the minimum amount of tuples to add or remove to make it consistent.

Examples of optimization problems

Examples

Maximum Matching. Given a set of tasks and a set of workers with preferences of workers on tasks, find an assignment for all tasks that maximizes satisfaction.

Set Cover. Given a set of people, each with fluency in various languages, find a group of people of minimum size who can speak all the languages.

Vertex Cover. In a city, find a set of minimum size of road intersections where to put street cameras such that all roads are covered.

Inconsistent Data Repair. Given a database inconsistent w.r.t. fixed integrity constraints, find the minimum amount of tuples to add or remove to make it consistent.

Influence Maximization. Given a social network with influence probabilities on edges, find the set of nodes to target to maximize the impact of a marketing campaign.

Examples of Computation Problems

Examples

Coloring Counting. Compute the number of ways to color a graph with 3 colors.

Examples of Computation Problems

Examples

Coloring Counting. Compute the number of ways to color a graph with 3 colors.

SQL Match Counting. Compute the number of distinct matches to a fixed SQL query:

```
SELECT COUNT(DISTINCT *)
FROM R NATURAL JOIN S NATURAL JOIN T
```

Examples of Computation Problems

Examples

Coloring Counting. Compute the number of ways to color a graph with 3 colors.

SQL Match Counting. Compute the number of distinct matches to a fixed SQL query:

```
SELECT COUNT(DISTINCT *)
FROM R NATURAL JOIN S NATURAL JOIN T
```

Navigational XPath Counting. Compute the number of matches of a fixed simple XPath expression (no functions, no equality):

```
count(//a[b/c]/d[e/f])
```

Examples of Computation Problems

Examples

Coloring Counting. Compute the number of ways to color a graph with 3 colors.

SQL Match Counting. Compute the number of distinct matches to a fixed SQL query:

```
SELECT COUNT(DISTINCT *)
FROM R NATURAL JOIN S NATURAL JOIN T
```

Navigational XPath Counting. Compute the number of matches of a fixed simple XPath expression (no functions, no equality):

```
count(//a[b/c]/d[e/f])
```

Probabilistic Query Evaluation. Compute the probability of a fixed SQL query over a database whose tuples are annotated with probabilities.

Intractability

- Most of these problems are intractable: unless $P = NP$, there is no polynomial-time algorithm to solve them, only algorithms exponential in the size of the data!

Intractability

- Most of these problems are intractable: unless $P = NP$, there is no polynomial-time algorithm to solve them, only algorithms exponential in the size of the data!
- Two classes of intractability discussed here: NP-hardness for optimization problems, #P-hardness for computation problems (latter implies former). See further.

Intractability

- Most of these problems are intractable: unless $P = NP$, there is no polynomial-time algorithm to solve them, only algorithms exponential in the size of the data!
- Two classes of intractability discussed here: NP-hardness for optimization problems, #P-hardness for computation problems (latter implies former). See further.

Polynomial-time	NP-hard	#P-hard
Max. Matching Nav. XPath Counting	Set Cover Vertex Cover Incons. Data Repair Influence Maximization	Coloring Counting SQL Match Counting Prob. Query Evaluation

Is this the end of it?

- Many real-world tasks require solving hard problems

Licence de droits d'usage

Is this the end of it?

- Many real-world tasks require solving hard problems
- Be persistent, do not stop because you have encountered an intractable problem!

Licence de droits d'usage

Is this the end of it?

- Many real-world tasks require solving hard problems
- Be persistent, do not stop because you have encountered an intractable problem!
- Different strategies:

Is this the end of it?

- Many real-world tasks require solving hard problems
- Be persistent, do not stop because you have encountered an intractable problem!
- Different strategies:
  - Find tractable subcases

Licence de droits d'usage

Is this the end of it?

- Many real-world tasks require solving hard problems
- Be persistent, do not stop because you have encountered an intractable problem!
- Different strategies:
  - Find tractable subcases
  - Find heuristic algorithms that are good enough in practice, though without any guarantee

Is this the end of it?

- Many real-world tasks require solving hard problems
- Be persistent, do not stop because you have encountered an intractable problem!
- Different strategies:
  - Find tractable subcases
  - Find heuristic algorithms that are good enough in practice, though without any guarantee
  - Find deterministic algorithms that provide a guaranteed approximation

Is this the end of it?

- Many real-world tasks require solving hard problems
- Be persistent, do not stop because you have encountered an intractable problem!
- Different strategies:
  - Find tractable subcases
  - Find heuristic algorithms that are good enough in practice, though without any guarantee
  - Find deterministic algorithms that provide a guaranteed approximation
  - Find randomized algorithms that provide a guaranteed approximation with high probability

Outline

Licence de droits d'usage

NP-hardness

- A decision (i.e., yes/no) problem is in NP if there exists a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess) that solves it

Licence de droits d'usage

NP-hardness

- A decision (i.e., yes/no) problem is in NP if there exists a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess) that solves it
- A problem X is NP-hard if it is at least as hard as any problem in NP: being able to solve X means you can solve any problem in NP with deterministic polynomial-time overhead

Licence de droits d'usage

NP-hardness

- A decision (i.e., yes/no) problem is in NP if there exists a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess) that solves it
- A problem *X* is NP-hard if it is at least as hard as any problem in NP: being able to solve *X* means you can solve any problem in NP with deterministic polynomial-time overhead
- NP-complete: decision problem both NP and NP-hard

Licence de droits d'usage

TELECOM ParisTech

NP-hardness

- A decision (i.e., yes/no) problem is in NP if there exists a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess) that solves it
- A problem X is NP-hard if it is at least as hard as any problem in NP: being able to solve X means you can solve any problem in NP with deterministic polynomial-time overhead
- NP-complete: decision problem both NP and NP-hard
- To prove NP-hardness of X, you show a polynomial-time reduction from an arbitrary problem Y known to be NP-hard to X: you take an instance of Y and show that if you know how to solve X, then you can solve Y with deterministic polynomial-time overhead

NP-hardness

- A decision (i.e., yes/no) problem is in NP if there exists a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess) that solves it
- A problem *X* is NP-hard if it is at least as hard as any problem in NP: being able to solve *X* means you can solve any problem in NP with deterministic polynomial-time overhead
- NP-complete: decision problem both NP and NP-hard
- To prove NP-hardness of *X*, you show a polynomial-time reduction from an arbitrary problem *Y* known to be NP-hard to *X*: you take an instance of *Y* and show that if you know how to solve *X*, then you can solve *Y* with deterministic polynomial-time overhead

Technical note: most definitions of NP-hardness for decision problems are a bit stricter because they are based on Karp many-one reductions. Important if you want to distinguish, e.g., NP-hardness vs coNP-hardness. For optimization/computation problems, it is irrelevant, so I use simpler Turing reductions.

#P-hardness

- A counting problem is in #P if it can be solved by counting the number of ways a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess, and you count the various ways to guess) can return “yes”

Licence de droits d’usage

#P-hardness

- A counting problem is in #P if it can be solved by counting the number of ways a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess, and you count the various ways to guess) can return “yes”
- A problem *X* is #P-hard if it is at least as hard as any problem in #P: being able to solve *X* means you can solve any problem in #P with deterministic polynomial-time overhead

Licence de droits d’usage

#P-hardness

- A counting problem is in #P if it can be solved by counting the number of ways a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess, and you count the various ways to guess) can return “yes”
- A problem X is #P-hard if it is at least as hard as any problem in #P: being able to solve X means you can solve any problem in #P with deterministic polynomial-time overhead
- To prove #P-hardness of X, you show a polynomial-time reduction from an arbitrary problem Y known to be #P-hard to X: you take an instance of Y and show that if you know how to solve X, then you can solve Y with deterministic polynomial-time overhead

#P-hardness

- A counting problem is in #P if it can be solved by counting the number of ways a nondeterministic polynomial-time algorithm (i.e., the algorithm is allowed to make a guess, and you count the various ways to guess) can return “yes”
- A problem X is #P-hard if it is at least as hard as any problem in #P: being able to solve X means you can solve any problem in #P with deterministic polynomial-time overhead
- To prove #P-hardness of X, you show a polynomial-time reduction from an arbitrary problem Y known to be #P-hard to X: you take an instance of Y and show that if you know how to solve X, then you can solve Y with deterministic polynomial-time overhead

Technical note: most definitions of #P-hardness for counting problems are a bit stricter because they are based on Karp many-one reductions. I use simpler Turing reductions, not much practical difference.

#P-hardness implies NP-hardness

Proof.

- Take a #P-hard problem X. Let us show it is NP-hard as well.

#P-hardness implies NP-hardness

Proof.

- Take a #P-hard problem X. Let us show it is NP-hard as well.
- Take an arbitrary NP-complete problem Y. There is a non-deterministic polynomial-time algorithm A that solves Y.

#P-hardness implies NP-hardness

Proof.

- Take a #P-hard problem X. Let us show it is NP-hard as well.
- Take an arbitrary NP-complete problem Y. There is a non-deterministic polynomial-time algorithm A that solves Y.
- Consider the problem Z that counts the number of ways A can return "yes". This is a #P problem.

#P-hardness implies NP-hardness

Proof.

- Take a #P-hard problem X. Let us show it is NP-hard as well.
- Take an arbitrary NP-complete problem Y. There is a non-deterministic polynomial-time algorithm A that solves Y.
- Consider the problem Z that counts the number of ways A can return “yes”. This is a #P problem.
- Thus Z reduces to X.

#P-hardness implies NP-hardness

Proof.

- Take a #P-hard problem X. Let us show it is NP-hard as well.
- Take an arbitrary NP-complete problem Y. There is a non-deterministic polynomial-time algorithm A that solves Y.
- Consider the problem Z that counts the number of ways A can return "yes". This is a #P problem.
- Thus Z reduces to X.
- But Y reduces to Z: use Z to count, and return "yes" iff the count is > 0.

□

Licence de droits d'usage

#P-hardness implies NP-hardness

Proof.

- Take a #P-hard problem X. Let us show it is NP-hard as well.
- Take an arbitrary NP-complete problem Y. There is a non-deterministic polynomial-time algorithm A that solves Y.
- Consider the problem Z that counts the number of ways A can return “yes”. This is a #P problem.
- Thus Z reduces to X.
- But Y reduces to Z: use Z to count, and return “yes” iff the count is > 0.
- Therefore Y reduces to X, and thus X is NP-hard.

□

Licence de droits d’usage

TELECOM ParisTech

Outline

Outline

Licence de droits d'usage

Additive (absolute) approximation

Let $\varphi : \mathbb{R} \to \mathbb{R}^+$.

Definition

An optimization algorithm A provides an additive φ-approximation for a problem P with optimal solution X^* if the solution X returned by A satisfies the condition of P and is such that

$$|f(X) - f(X^*)| \leqslant \varphi(f(X^*))$$

Additive (absolute) approximation

Let $\varphi : \mathbb{R} \to \mathbb{R}^+$.

Definition

An optimization algorithm A provides an additive φ-approximation for a problem P with optimal solution X^* if the solution X returned by A satisfies the condition of P and is such that

$$|f(X) - f(X^*)| \leqslant \varphi(f(X^*))$$

Definition

A computation algorithm A provides an additive φ-approximation for a problem P with actual solution v^* if the value v returned by A is such that

$$|v - v^*| \leqslant \varphi(v^*)$$

Multiplicative (relative) approximation

Let $\varphi : \mathbb{R} \to \mathbb{R}^+$.

Definition

An optimization algorithm A provides a multiplicative φ-approximation for a problem P with optimal solution X^* if the solution X returned by A satisfies the condition of P and is such that

$$|f(X)| \leqslant \varphi(f(X^*))|f(X)^*| \qquad \text{if } P \text{ is a minimization problem}$$
$$|f(X)| \geqslant \varphi(f(X^*))|f(X)^*| \qquad \text{if } P \text{ is a maximization problem}$$

Multiplicative (relative) approximation

Let $\varphi : \mathbb{R} \to \mathbb{R}^+$.

Definition

An optimization algorithm A provides a multiplicative φ-approximation for a problem P with optimal solution X^* if the solution X returned by A satisfies the condition of P and is such that

$$|f(X)| \leqslant \varphi(f(X^*))|f(X)^*| \qquad \text{if } P \text{ is a minimization problem}$$

$$|f(X)| \geqslant \varphi(f(X^*))|f(X)^*| \qquad \text{if } P \text{ is a maximization problem}$$

Definition (attention, inconsistent notation!)

A computation algorithm A provides a multiplicative φ-approximation for a problem P with actual solution v^* if the value v returned by A is such that

$$(1 - \varphi(v^*))|v^*| \leqslant |v| \leqslant (1 + \varphi(v^*))|v^*|$$

APX

- We want the approximation algorithms to be polynomial-time

Licence de droits d'usage

APX

- We want the approximation algorithms to be polynomial-time
- Ideally, φ is constant

APX

- We want the approximation algorithms to be polynomial-time
- Ideally, φ is constant
- For a constant φ, multiplicative approximation is better than additive approximation

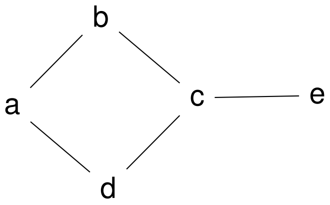
APX

- We want the approximation algorithms to be polynomial-time
- Ideally, φ is constant
- For a constant φ, multiplicative approximation is better than additive approximation
- Additive approximation is thus rarely used, “approximation algorithm” usually means multiplicative

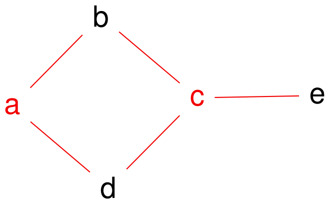
APX

- We want the approximation algorithms to be polynomial-time
- Ideally, φ is constant
- For a constant φ, multiplicative approximation is better than additive approximation
- Additive approximation is thus rarely used, “approximation algorithm” usually means multiplicative
- APX: class of optimization problems that have a polynomial-time multiplicative approximation algorithm with constant φ

Vertex Cover is in APX

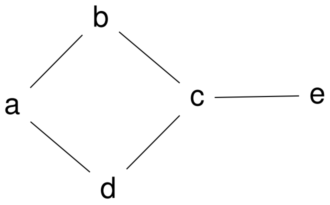


Vertex Cover is in APX



Optimal: $\{a, c\}$, size 2

Vertex Cover is in APX

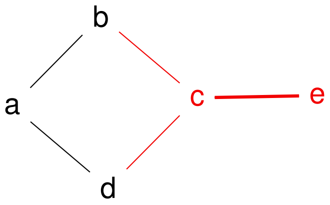


Optimal: $\{a, c\}$, size 2

Approximation algorithm:

- Choose an arbitrary edge not covered
- Add both end points to the cover
- Repeat until all edges are covered

Vertex Cover is in APX

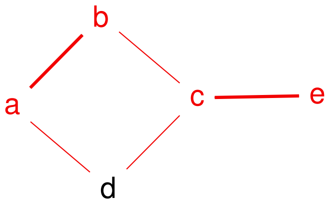


Optimal: $\{a, c\}$, size 2

Approximation algorithm:

- Choose an arbitrary edge not covered
- Add both end points to the cover
- Repeat until all edges are covered

Vertex Cover is in APX

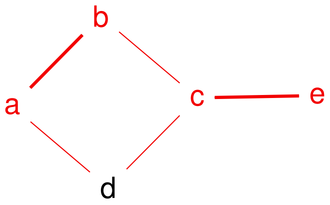


Optimal: $\{a, c\}$, size 2

Approximation algorithm:

- Choose an arbitrary edge not covered
- Add both end points to the cover
- Repeat until all edges are covered

Vertex Cover is in APX



Optimal: $\{a, c\}$, size 2

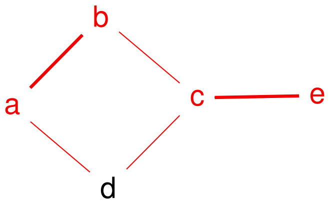
Approximated: $\{a, b, c, e\}$, size 4

Approximation algorithm:

- Choose an arbitrary edge not covered
- Add both end points to the cover
- Repeat until all edges are covered

Licence de droits d'usage

Vertex Cover is in APX



Optimal: $\{a, c\}$, size 2

Approximated: $\{a, b, c, e\}$, size 4

Approximation algorithm:

- Choose an arbitrary edge not covered
- Add both end points to the cover
- Repeat until all edges are covered

Multiplicative 2-approximation! (twice as many nodes in the approximated cover as edges chosen, each of this edge need to be covered)

Other examples

- Set Cover has a $(\ln n + O(1))$-approximation algorithm [Chv79] but is not in APX [LY94]
- Inconsistent Data Repair is in APX (but the constant depends on the dependencies) [KL09]
- Influence Maximization is in APX; it has a $(1 - 1/\mathrm{e} - \varepsilon)$-approximation algorithm for any ε (slightly better than 63%) [KKT03]

Inapproximability results

- It is also possible to show that some problem is not φ-approximable (we assume $\mathrm{P} \neq \mathrm{NP}$ in this slide)

Inapproximability results

- It is also possible to show that some problem is not φ-approximable (we assume P $\neq$ NP in this slide)
- Vertex Cover is not 1.3606-approximable [DS05] (!)

Licence de droits d'usage

Inapproximability results

- It is also possible to show that some problem is not φ-approximable (we assume P $\neq$ NP in this slide)
- Vertex Cover is not 1.3606-approximable [DS05] (!)
- Set Cover is not $(\ln(n) - o(\ln n))$-approximable [DS14]

Inapproximability results

- It is also possible to show that some problem is not φ-approximable (we assume P $\neq$ NP in this slide)
- Vertex Cover is not 1.3606-approximable [DS05] (!)
- Set Cover is not $(\ln(n) - o(\ln n))$-approximable [DS14]
- This kind of results is usually much more difficult to obtain than approximation algorithms

How to find an approximation algorithm?

- From scratch, by exploiting the structure of the problem (as we did with Vertex Cover)

How to find an approximation algorithm?

- From scratch, by exploiting the structure of the problem (as we did with Vertex Cover)
- By exploiting approximation-preserving reductions between a problem and an approximable problem (in both directions); various notions of approximation-preserving, arbitrary reductions don't work

Outline

Licence de droits d'usage

From APX to FPTAS

- APX: polynomial-time c-approximation for some fixed constant c

From APX to FPTAS

- APX: polynomial-time c-approximation for some fixed constant c
- Useful, but would be better if we could have a c arbitrarily close to 1

From APX to FPTAS

- APX: polynomial-time c-approximation for some fixed constant c
- Useful, but would be better if we could have a c arbitrarily close to 1
- PTAS (Polynomial-Time Approximation Scheme): there exists a polynomial-time $(1+\varepsilon)$-approximation for any $\varepsilon > 0$ (for a minimization problem); $(1-\varepsilon)$-approximation for a maximization problem; ε-approximation for a computation problem

From APX to FPTAS

- APX: polynomial-time c-approximation for some fixed constant c
- Useful, but would be better if we could have a c arbitrarily close to 1
- PTAS (Polynomial-Time Approximation Scheme): there exists a polynomial-time $(1+\varepsilon)$-approximation for any $\varepsilon > 0$ (for a minimization problem); $(1-\varepsilon)$-approximation for a maximization problem; ε-approximation for a computation problem
- Great, but these approximations may become more and more difficult to find as ε nears 0

From APX to FPTAS

- APX: polynomial-time c-approximation for some fixed constant c
- Useful, but would be better if we could have a c arbitrarily close to 1
- PTAS (Polynomial-Time Approximation Scheme): there exists a polynomial-time $(1+\varepsilon)$-approximation for any $\varepsilon > 0$ (for a minimization problem); $(1-\varepsilon)$-approximation for a maximization problem; ε-approximation for a computation problem
- Great, but these approximations may become more and more difficult to find as ε nears 0
- FPTAS (Fully Polynomial-Time Approximation Scheme): PTAS whose overall complexity depends polynomially in $1/\varepsilon$

Problems with a FPTAS?

- Fairly rare!

Licence de droits d'usage

Problems with a FPTAS?

- Fairly rare!
- Neither Vertex Cover, nor Inconsistent Data Repair [KL09], nor Influence Maximization [KKT03] have an FPTAS ☹ (unless $P = NP$)

Problems with a FPTAS?

- Fairly rare!
- Neither Vertex Cover, nor Inconsistent Data Repair [KL09], nor Influence Maximization [KKT03] have an FPTAS ☹ (unless $P = NP$)
- There are still some problems for which there are FPTAS:

Problems with a FPTAS?

- Fairly rare!
- Neither Vertex Cover, nor Inconsistent Data Repair [KL09], nor Influence Maximization [KKT03] have an FPTAS ☹ (unless P = NP)
- There are still some problems for which there are FPTAS:

Example

Problems with a FPTAS?

- Fairly rare!
- Neither Vertex Cover, nor Inconsistent Data Repair [KL09], nor Influence Maximization [KKT03] have an FPTAS ☹ (unless P = NP)
- There are still some problems for which there are FPTAS:

Example

Knapsack. Given a collection of items, each with a weight and a volume, find a subset of items of maximum volume whose total weight does not exceed some fixed limit

Problems with a FPTAS?

- Fairly rare!
- Neither Vertex Cover, nor Inconsistent Data Repair [KL09], nor Influence Maximization [KKT03] have an FPTAS ☹ (unless $P = NP$)
- There are still some problems for which there are FPTAS:

Example

Knapsack. Given a collection of items, each with a weight and a volume, find a subset of items of maximum volume whose total weight does not exceed some fixed limit

- Knapsack is an NP-hard problem, but there exists an FPTAS

Licence de droits d'usage

Outline

Introduction

Intractable Classes

Deterministic Approximations

Randomized Approximations

- Generalities
- Monte-Carlo Sampling
- FPRAS

Conclusion

Outline

Randomized approximations

To simplify, only talk about computation problems. $\varphi : \mathbb{R} \to \mathbb{R}^+$, $\delta > 0$.

Definition

A computation algorithm A provides a randomized additive (φ, δ)-approximation for a problem P with actual solution v^* if the value v returned by A is such that

$$|v - v^*| \leqslant \varphi(v^*) \quad \text{with probability} \geqslant 1 - \delta$$

Licence de droits d'usage

Randomized approximations

To simplify, only talk about computation problems. $\varphi : \mathbb{R} \to \mathbb{R}^+$, $\delta > 0$.

Definition

A computation algorithm A provides a randomized additive (φ, δ)-approximation for a problem P with actual solution v^* if the value v returned by A is such that

$$|v - v^*| \leqslant \varphi(v^*) \quad \text{with probability} \geqslant 1 - \delta$$

Definition

A computation algorithm A provides a randomized multiplicative (φ, δ)-approximation for a problem P with actual solution v^* if the value v returned by A is such that

$$(1 - \varphi(v^*))|v^*| \leqslant |v| \leqslant (1 + \varphi(v^*))|v^*| \quad \text{with probability} \geqslant 1 - \delta$$

Hoeffding's Inequality

Let $X_1, \ldots, X_n$ be n independent random variables, each within the interval $[a, b]$, and $\bar{X} = \frac{1}{n} \sum_i X_i$ the empirical mean.

Licence de droits d'usage

Hoeffding's Inequality

Let $X_1, \dots, X_n$ be n independent random variables, each within the interval $[a, b]$, and $\bar{X} = \frac{1}{n} \sum_i X_i$ the empirical mean.

We have [Hoe63]:

$$\Pr\left[\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geqslant \varepsilon\right] \leqslant 2\mathrm{e}^{\frac{-2n\varepsilon^2}{(b-a)^2}}$$

Hoeffding's Inequality

Let $X_1, \dots, X_n$ be n independent random variables, each within the interval $[a, b]$, and $\bar{X} = \frac{1}{n} \sum_i X_i$ the empirical mean.

We have [Hoe63]:

$$\Pr\left[\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geqslant \varepsilon\right] \leqslant 2\mathrm{e}^{\frac{-2n\varepsilon^2}{(b-a)^2}}$$

In other words, we know that $\Pr\left[\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geqslant \varepsilon\right] \leqslant \delta$ as long as:

Hoeffding's Inequality

Let $X_1, \ldots, X_n$ be n independent random variables, each within the interval $[a, b]$, and $\bar{X} = \frac{1}{n} \sum_i X_i$ the empirical mean.

We have [Hoe63]:

$$\Pr\left[\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geqslant \varepsilon\right] \leqslant 2\mathrm{e}^{\frac{-2n\varepsilon^2}{(b-a)^2}}$$

In other words, we know that $\Pr\left[\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geqslant \varepsilon\right] \leqslant \delta$ as long as:

$$n \geqslant \frac{(b-a)^2}{2\varepsilon^2} \ln \frac{1}{\delta}$$

Hoeffding's Inequality

Let $X_1, \ldots, X_n$ be n independent random variables, each within the interval $[a, b]$, and $\bar{X} = \frac{1}{n}\sum_i X_i$ the empirical mean.

We have [Hoe63]:

$$\Pr\left[\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geqslant \varepsilon\right] \leqslant 2\mathrm{e}^{\frac{-2n\varepsilon^2}{(b-a)^2}}$$

In other words, we know that $\Pr\left[\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geqslant \varepsilon\right] \leqslant \delta$ as long as:

$$n \geqslant \frac{(b-a)^2}{2\varepsilon^2} \ln \frac{1}{\delta}$$

Often too conservative!

Outline

Application to Polling

- Poll of n persons in a country of m inhabitants

Licence de droits d'usage

Application to Polling

- Poll of n persons in a country of m inhabitants
- Every person i is asked if they prefer politician A or politician B. We note $X_i = 0$ if they prefer A, 1 otherwise

Licence de droits d'usage

Application to Polling

- Poll of n persons in a country of m inhabitants
- Every person i is asked if they prefer politician A or politician B. We note $X_i = 0$ if they prefer A, 1 otherwise
- We are interested in predicting the result of an election between A and B; $\mathbb{E}[\bar{X}]$ is the expected proportion of votes for B

Application to Polling

- Poll of n persons in a country of m inhabitants
- Every person i is asked if they prefer politician A or politician B. We note $X_i = 0$ if they prefer A, 1 otherwise
- We are interested in predicting the result of an election between A and B; $\mathbb{E}[\bar{X}]$ is the expected proportion of votes for B
- We want a margin of error of $\varepsilon = 2\%$, and a probabilistic guarantee of $1 - \delta = 95\%$

Application to Polling

- Poll of n persons in a country of m inhabitants
- Every person i is asked if they prefer politician A or politician B. We note $X_i = 0$ if they prefer A, 1 otherwise
- We are interested in predicting the result of an election between A and B; $\mathbb{E}[\bar{X}]$ is the expected proportion of votes for B
- We want a margin of error of $\varepsilon = 2\%$, and a probabilistic guarantee of $1 - \delta = 95\%$
- So we just need by Hoeffding's inequality:

$$n \geqslant \frac{1}{2\varepsilon^2} \ln \frac{1}{\delta} \geqslant 3745$$

Application to Polling

- Poll of n persons in a country of m inhabitants
- Every person i is asked if they prefer politician A or politician B. We note $X_i = 0$ if they prefer A, 1 otherwise
- We are interested in predicting the result of an election between A and B; $\mathbb{E}[\bar{X}]$ is the expected proportion of votes for B
- We want a margin of error of $\varepsilon = 2\%$, and a probabilistic guarantee of $1 - \delta = 95\%$
- So we just need by Hoeffding's inequality:

$$n \geqslant \frac{1}{2\varepsilon^2} \ln \frac{1}{\delta} \geqslant 3745$$

- This is completely independent of m!

Monte-Carlo Sampling

- Assumptions:

Examples

Monte-Carlo Sampling

- Assumptions:
  - We can sample in polynomial-time from a population

Examples

Monte-Carlo Sampling

- Assumptions:
  - We can sample in polynomial-time from a population
  - Given a sample, we can evaluate a certain quantity in polynomial-time

Examples

Monte-Carlo Sampling

- Assumptions:
  - We can sample in polynomial-time from a population
  - Given a sample, we can evaluate a certain quantity in polynomial-time
- Then we can compute the expected mean of that quantity with a polynomial-time randomized additive (ε, δ)-approximation algorithm for arbitrary $\varepsilon > 0$, $\delta > 0$

Examples

Monte-Carlo Sampling

- Assumptions:
  - We can sample in polynomial-time from a population
  - Given a sample, we can evaluate a certain quantity in polynomial-time
- Then we can compute the expected mean of that quantity with a polynomial-time randomized additive (ε, δ)-approximation algorithm for arbitrary $\varepsilon > 0$, $\delta > 0$
- Direct application of Hoeffding's inequality, can be used to obtain the required number of samples

Examples

Monte-Carlo Sampling

- Assumptions:
  - We can sample in polynomial-time from a population
  - Given a sample, we can evaluate a certain quantity in polynomial-time
- Then we can compute the expected mean of that quantity with a polynomial-time randomized additive (ε, δ)-approximation algorithm for arbitrary $\varepsilon > 0$, $\delta > 0$
- Direct application of Hoeffding's inequality, can be used to obtain the required number of samples

Examples

Polling

Monte-Carlo Sampling

- Assumptions:
  - We can sample in polynomial-time from a population
  - Given a sample, we can evaluate a certain quantity in polynomial-time
- Then we can compute the expected mean of that quantity with a polynomial-time randomized additive (ε, δ)-approximation algorithm for arbitrary $\varepsilon > 0$, $\delta > 0$
- Direct application of Hoeffding's inequality, can be used to obtain the required number of samples

Examples

Polling

Computation of π

Monte-Carlo Sampling

- Assumptions:
  - We can sample in polynomial-time from a population
  - Given a sample, we can evaluate a certain quantity in polynomial-time
- Then we can compute the expected mean of that quantity with a polynomial-time randomized additive (ε, δ)-approximation algorithm for arbitrary $\varepsilon > 0$, $\delta > 0$
- Direct application of Hoeffding's inequality, can be used to obtain the required number of samples

Examples

Polling

Computation of π

Probabilistic Query Evaluation

Outline

FPRAS

- PRAS (Polynomial-time Randomized Approximation Scheme): there exists a polynomial-time randomized $(\varepsilon, 1/3)$-approximation for any $\varepsilon > 0$

FPRAS

- PRAS (Polynomial-time Randomized Approximation Scheme): there exists a polynomial-time randomized $(\varepsilon, 1/3)$-approximation for any $\varepsilon > 0$
- $1/3$ is irrelevant here; from that, we can obtain an (ε, δ)-approximation for the same ε and arbitrary δ by simply repeating the algorithm

FPRAS

- PRAS (Polynomial-time Randomized Approximation Scheme): there exists a polynomial-time randomized $(\varepsilon, 1/3)$-approximation for any $\varepsilon > 0$
- $1/3$ is irrelevant here; from that, we can obtain an (ε, δ)-approximation for the same ε and arbitrary δ by simply repeating the algorithm
- Great, but these approximations may become more and more difficult to find as ε nears 0

FPRAS

- PRAS (Polynomial-time Randomized Approximation Scheme): there exists a polynomial-time randomized $(\varepsilon, 1/3)$-approximation for any $\varepsilon > 0$
- $1/3$ is irrelevant here; from that, we can obtain an (ε, δ)-approximation for the same ε and arbitrary δ by simply repeating the algorithm
- Great, but these approximations may become more and more difficult to find as ε nears 0
- FPRAS (Fully Polynomial-time Randomized Approximation Scheme): PRAS whose overall complexity depends polynomially in $1/\varepsilon$

FPRAS for disjunctions [KLM89, KKS09]

- $E_1, \ldots, E_m$ sequence of events in a probability space

FPRAS for disjunctions [KLM89, KKS09]

- $E_1, \ldots, E_m$ sequence of events in a probability space
- Assumptions: For each E_i, we can efficiently (in polynomial-time):

FPRAS for disjunctions [KLM89, KKS09]

- $E_1, \dots, E_m$ sequence of events in a probability space
- Assumptions: For each E_i, we can efficiently (in polynomial-time):
  - Compute $\Pr(E_i)$

FPRAS for disjunctions [KLM89, KKS09]

- $E_1, \ldots, E_m$ sequence of events in a probability space
- Assumptions: For each E_i, we can efficiently (in polynomial-time):
  - Compute $\Pr(E_i)$
  - Test whether E_i is true in a given random sample

Licence de droits d'usage

FPRAS for disjunctions [KLM89, KKS09]

- $E_1, \ldots, E_m$ sequence of events in a probability space
- Assumptions: For each E_i, we can efficiently (in polynomial-time):
  - Compute $\Pr(E_i)$
  - Test whether E_i is true in a given random sample
  - Sample from the subspace conditioned on E_i

Licence de droits d'usage

FPRAS for disjunctions [KLM89, KKS09]

- $E_1, \ldots, E_m$ sequence of events in a probability space
- Assumptions: For each E_i, we can efficiently (in polynomial-time):
  - Compute $\Pr(E_i)$
  - Test whether E_i is true in a given random sample
  - Sample from the subspace conditioned on E_i
- Then there exists a FPRAS to compute $\Pr\left(\bigvee_{i=1}^{m} E_i\right)$

FPRAS for disjunctions [KLM89, KKS09]

- $E_1, \ldots, E_m$ sequence of events in a probability space
- Assumptions: For each E_i, we can efficiently (in polynomial-time):
  - Compute $\Pr(E_i)$
  - Test whether E_i is true in a given random sample
  - Sample from the subspace conditioned on E_i
- Then there exists a FPRAS to compute $\Pr\left(\bigvee_{i=1}^{m} E_i\right)$
- See `http://webcourse.cs.technion.ac.il/236605/Spring2015/ho/WCFiles/L9%20-%20QA%20in%20PDBs.pdf` for detailed explanations

Application

- Probabilistic Query Evaluation has a FPRAS when:

Application

- Probabilistic Query Evaluation has a FPRAS when:
  - The query is a union of conjunctive queries (i.e., a `UNION` of `SELECT ... FROM ... WHERE` without any embedded queries)

Application

- Probabilistic Query Evaluation has a FPRAS when:
  - The query is a union of conjunctive queries (i.e., a `UNION` of `SELECT ... FROM ... WHERE` without any embedded queries)
  - Probability annotations on tuples are all elementary (no complex correlations)

Application

- Probabilistic Query Evaluation has a FPRAS when:
  - The query is a union of conjunctive queries (i.e., a `UNION` of `SELECT ... FROM ... WHERE` without any embedded queries)
  - Probability annotations on tuples are all elementary (no complex correlations)
- Simply compute the lineage of the query on the database

Application

- Probabilistic Query Evaluation has a FPRAS when:
  - The query is a union of conjunctive queries (i.e., a `UNION` of `SELECT ... FROM ... WHERE` without any embedded queries)
  - Probability annotations on tuples are all elementary (no complex correlations)
- Simply compute the lineage of the query on the database
- Because of the shape of the query, the lineage is a disjunction of conjunctions

Application

- Probabilistic Query Evaluation has a FPRAS when:
  - The query is a union of conjunctive queries (i.e., a `UNION` of `SELECT ... FROM ... WHERE` without any embedded queries)
  - Probability annotations on tuples are all elementary (no complex correlations)
- Simply compute the lineage of the query on the database
- Because of the shape of the query, the lineage is a disjunction of conjunctions
- All three conditions for the existence of the FPRAS are satisfied

Application

- Probabilistic Query Evaluation has a FPRAS when:
  - The query is a union of conjunctive queries (i.e., a `UNION` of `SELECT ... FROM ... WHERE` without any embedded queries)
  - Probability annotations on tuples are all elementary (no complex correlations)
- Simply compute the lineage of the query on the database
- Because of the shape of the query, the lineage is a disjunction of conjunctions
- All three conditions for the existence of the FPRAS are satisfied
- Extends to SQL Match Counting, provided there is no projection (because count is proportional to probability)

Application

- Probabilistic Query Evaluation has a FPRAS when:
  - The query is a union of conjunctive queries (i.e., a `UNION` of `SELECT ... FROM ... WHERE` without any embedded queries)
  - Probability annotations on tuples are all elementary (no complex correlations)
- Simply compute the lineage of the query on the database
- Because of the shape of the query, the lineage is a disjunction of conjunctions
- All three conditions for the existence of the FPRAS are satisfied
- Extends to SQL Match Counting, provided there is no projection (because count is proportional to probability)
- But one can show there is no FPRAS for arbitary SQL queries

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\; R(x) \wedge S(x, y) \wedge T(y)$$

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\; R(x) \wedge S(x, y) \wedge T(y)$$

R	
a	f_1
b	f_2
c	f_3

S		
a	a	g_1
b	v	g_2
b	w	g_3

T	
v	h_1
w	h_2
b	h_3

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\; R(x) \wedge S(x, y) \wedge T(y)$$

R	
a	f_1
b	f_2
c	f_3

S		
a	a	g_1
b	v	g_2
b	w	g_3

T	
v	h_1
w	h_2
b	h_3

$\rightarrow$ Lineage:

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\ R(x) \land S(x, y) \land T(y)$$

R	
a	f_1
b	f_2
c	f_3

S		
a	a	g_1
b	v	g_2
b	w	g_3

T	
v	h_1
w	h_2
b	h_3

→ Lineage: $f_2 \land$

Licence de droits d'usage

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\; R(x) \wedge S(x, y) \wedge T(y)$$

R	
a	f_1
b	f_2
c	f_3

S		
a	a	g_1
b	v	g_2
b	w	g_3

T	
v	h_1
w	h_2
b	h_3

→ Lineage: $f_2 \wedge ($

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\; R(x) \land S(x, y) \land T(y)$$

R	
a	f_1
b	f_2
c	f_3

S		
a	a	g_1
b	v	g_2
b	w	g_3

T	
v	h_1
w	h_2
b	h_3

$\rightarrow$ Lineage: $f_2 \land ((g_2 \land h_1)$

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\ R(x) \wedge S(x, y) \wedge T(y)$$

R	
a	f_1
b	f_2
c	f_3

S		
a	a	g_1
b	v	g_2
b	w	g_3

T	
v	h_1
w	h_2
b	h_3

→ Lineage: $f_2 \wedge ((g_2 \wedge h_1) \vee (g_3 \wedge h_2))$

Technical tool: lineages

(slide from A. Amarilli)

The lineage of a query q on an instance I:

- Boolean function φ whose variables are the facts of I
- A subinstance of I satisfies q iff φ is true for that valuation

```
SELECT DISTINCT 1 -- Boolean query
FROM R NATURAL JOIN ON S NATURAL JOIN ON T
```

$$q : \exists x\, y\; R(x) \wedge S(x,y) \wedge T(y)$$

R	
a	f_1
b	f_2
c	f_3

S		
a	a	g_1
b	v	g_2
b	w	g_3

T	
v	h_1
w	h_2
b	h_3

→ Lineage: $f_2 \wedge ((g_2 \wedge h_1) \vee (g_3 \wedge h_2)) = (f_2 \wedge g_2 \wedge h_1) \vee (f_2 \wedge g_3 \wedge h_2)$

Outline

Introduction

Intractable Classes

Deterministic Approximations

Randomized Approximations

Conclusion

Licence de droits d'usage

In brief

- Many real-world problems are intractable

In brief

- Many real-world problems are intractable
- Most can be approximated to some extent

In brief

- Many real-world problems are intractable
- Most can be approximated to some extent
- Some can be approximated within a constant factor (APX)

In brief

- Many real-world problems are intractable
- Most can be approximated to some extent
- Some can be approximated within a constant factor (APX)
- Few have a FPTAS, which makes approximation fully efficient and practical

In brief

- Many real-world problems are intractable
- Most can be approximated to some extent
- Some can be approximated within a constant factor (APX)
- Few have a FPTAS, which makes approximation fully efficient and practical
- Monte-Carlo sampling is a very general technique that can be used to obtain randomized additive approximations within an arbitrary constant

In brief

- Many real-world problems are intractable
- Most can be approximated to some extent
- Some can be approximated within a constant factor (APX)
- Few have a FPTAS, which makes approximation fully efficient and practical
- Monte-Carlo sampling is a very general technique that can be used to obtain randomized additive approximations within an arbitrary constant
- But additive approximations are generally not as good as multiplicative approximations

In brief

- Many real-world problems are intractable
- Most can be approximated to some extent
- Some can be approximated within a constant factor (APX)
- Few have a FPTAS, which makes approximation fully efficient and practical
- Monte-Carlo sampling is a very general technique that can be used to obtain randomized additive approximations within an arbitrary constant
- But additive approximations are generally not as good as multiplicative approximations
- Some problems have a FPRAS, which makes randomized approximation fully efficient and practical

To go further

- Claire Mathieu's MOOC on approximation algorithms
  - https://www.coursera.org/learn/approximation-algorithms-part-1
  - https://www.coursera.org/learn/approximation-algorithms-part-2
- Basics of computational complexity [Pap07]
- Textbook on approximation algorithms [Vaz01]
- How to design approximation algorithms? [WS11]
- Probabilistic databases [SORK11]

To go further

- Claire Mathieu's MOOC on approximation algorithms
  - https://www.coursera.org/learn/approximation-algorithms-part-1
  - https://www.coursera.org/learn/approximation-algorithms-part-2
- Basics of computational complexity [Pap07]
- Textbook on approximation algorithms [Vaz01]
- How to design approximation algorithms? [WS11]
- Probabilistic databases [SORK11]

Merci.

Licence de droits d'usage

References I

[Chv79] Vasek Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[DS05] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):16–30, 1963.

[KKS09] Benny Kimelfeld, Yuri Kosharovsky, and Yehoshua Sagiv. Query evaluation over probabilistic XML. *VLDB J.*, 18(5):1117–1140, 2009.

References II

[KKT03] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In Lise Getoor, Ted E. Senator, Pedro M. Domingos, and Christos Faloutsos, editors, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 137–146. ACM, 2003.

[KL09] Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In Ronald Fagin, editor, *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, volume 361 of *ACM International Conference Proceeding Series*, pages 53–62. ACM, 2009.

References III

[KLM89] Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[Pap07] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.

[SORK11] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Licence de droits d'usage

Contexte public } avec modifications

Par le téléchargement ou la consultation de ce document, l'utilisateur accepte la licence d'utilisation qui y est attachée, telle que détaillée dans les dispositions suivantes, et s'engage à la respecter intégralement.

La licence confère à l'utilisateur un droit d'usage sur le document consulté ou téléchargé, totalement ou en partie, dans les conditions définies ci-après et à l'exclusion expresse de toute utilisation commerciale.
Le droit d'usage défini par la licence autorise un usage à destination de tout public qui comprend :
– le droit de reproduire tout ou partie du document sur support informatique ou papier,
– le droit de diffuser tout ou partie du document au public sur support papier ou informatique, y compris par la mise à la disposition du public sur un réseau numérique,
– le droit de modifier la forme ou la présentation du document,
– le droit d'intégrer tout ou partie du document dans un document composite et de le diffuser dans ce nouveau document, à condition que :
– L'auteur soit informé.

Les mentions relatives à la source du document et/ou à son auteur doivent être conservées dans leur intégralité.

Le droit d'usage défini par la licence est personnel et non exclusif.
Tout autre usage que ceux prévus par la licence est soumis à autorisation préalable et expresse de l'auteur : sitepedago@telecom-paristech.fr