Pulse Survey

Today we will cover:

Lecture 3: Intractability and Reductions

Today we will cover:

- Intractable problems.

Today we will cover:

- Intractable problems.
- Optimization vs. decision problems.

Lecture 3: Intractability and Reductions

Today we will cover:

- Intractable problems.
- Optimization vs. decision problems.
- Reduction between problems.

Today we will cover:

- Intractable problems.
- Optimization vs. decision problems.
- Reduction between problems.

- **Reading material:** "Computers and Intractability", sec. 1.3.: Polynomial Time Algorithms and Intractable Problems.

Intractable Problems

Terminology: *Intractable* problems are those that solving them that require super-polynomial amount of time.

Intractable Problems

Terminology: *Intractable* problems are those that solving them that require super-polynomial amount of time.

We have two categories of them:

- Undecidable problems
- Decidable problems

Undecidable Intractable Problems

Undecidable Intractable Problems

- The earliest intractable problem was Turing's halting problem. More generally, Turing showed that there existed problems that are undecidable, no algorithm can ever solve them.

Undecidable Intractable Problems

- The earliest intractable problem was Turing's halting problem. More generally, Turing showed that there existed problems that are undecidable, no algorithm can ever solve them.
- Other problems of this sort are Hilbert's tenth problem: solvability of polynomial equations in integers.

Undecidable Intractable Problems

- The earliest intractable problem was Turing's halting problem. More generally, Turing showed that there existed problems that are undecidable, no algorithm can ever solve them.
- Other problems of this sort are Hilbert's tenth problem: solvability of polynomial equations in integers.
- Various plane tiling problem.

Decidable Intractable Problems

- The Time Hierarchy theorem implies that there are problems that require super-polynomial runtime.

Decidable Intractable Problems

- The Time Hierarchy theorem implies that there are problems that require super-polynomial runtime.
- The simplest examples are EXPTIME-complete problems, such as the bounded halting problem:

Decidable Intractable Problems

- The Time Hierarchy theorem implies that there are problems that require super-polynomial runtime.
- The simplest examples are EXPTIME-complete problems, such as the bounded halting problem:
  - Given a TM M and an integer k, return 1 if M terminates within k steps and 0 otherwise.

Types of Problems

Types of Problems

Optimization problems

- Find a good solution for an instance.

Types of Problems

Optimization problems

- Find a good solution for an instance.

find the length of the shortest path from A to B

Types of Problems

Optimization problems

- Find a good solution for an instance.

find the length of the shortest path from A to B

find the length of the shortest simple circuit in this graph

Types of Problems

Optimization problems

- Find a good solution for an instance.

find the length of the shortest path from A to B

find the length of the shortest simple circuit in this graph

Decision problems

- Answer is yes or no.

Types of Problems

Optimization problems

- Find a good solution for an instance.

find the length of the shortest path from A to B

find the length of the shortest simple circuit in this graph

Decision problems

- Answer is yes or no.

Is n a prime number?

Types of Problems

Optimization problems

- Find a good solution for an instance.

find the length of the shortest path from A to B

find the length of the shortest simple circuit in this graph

Decision problems

- Answer is yes or no.

Is n a prime number?

Is there a simple circuit of weight $\leq d$?

Types of Problems

Optimization problems

- Find a good solution for an instance.

find the length of the shortest path from A to B

find the length of the shortest simple circuit in this graph

Decision problems

- Answer is yes or no.

Is n a prime number?

Is there a simple circuit of weight $\leq d$?

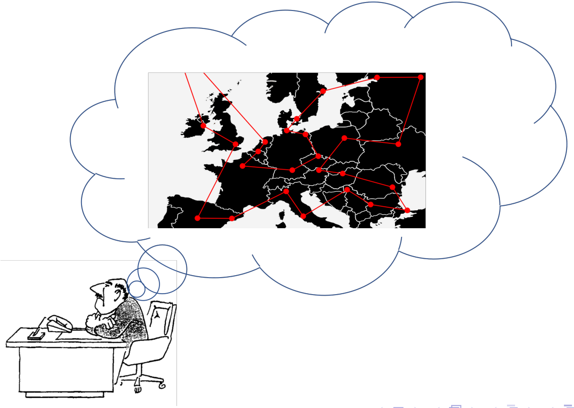
Does M halts on input w?

Travelling Salesman Problem (TSP)

- Instance: an integer weighted graph (G, w).

Travelling Salesman Problem (TSP)

- Instance: an integer weighted graph (G, w).
- Output: The length of a shortest weighted path traversing all nodes exactly once.



Travelling Salesman Decision Problem (TSDP)

- Instance: an integer weighted graph (G, w) and a non-negative integer d.

Travelling Salesman Decision Problem (TSDP)

- Instance: an integer weighted graph (G, w) and a non-negative integer d.
- Yes-instance: if there is a path covering all nodes of G of weight no more than d.

Travelling Salesman Decision Problem (TSDP)

- Instance: an integer weighted graph (G, w) and a non-negative integer d.
- Yes-instance: if there is a path covering all nodes of G of weight no more than d.
- No-instance: otherwise.

TMs and Optimization Problems

Let M be a TM, let $w \in \Sigma^*$, and denote:

$$f_M(w) = \begin{cases} \text{Contents of } M\text{'s tape after it terminates} & \text{If } M \text{ halts on } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

TMs and Optimization Problems

Let M be a TM, let $w \in \Sigma^*$, and denote:

$$f_M(w) = \begin{cases} \text{Contents of } M\text{'s tape after it terminates} & \text{If } M \text{ halts on } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

E.g., for TSP, if whenever

$$\text{Input : Encoding of weighted graph}$$

TMs and Optimization Problems

Let M be a TM, let $w \in \Sigma^*$, and denote:

$$f_M(w) = \begin{cases} \text{Contents of } M\text{'s tape after it terminates} & \text{If } M \text{ halts on } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

E.g., for TSP, if whenever

$$\text{Input : Encoding of weighted graph}$$

$$\text{Outcome : } f_M(w) \text{ is a shortest circuit}$$

then M solves TSP.

TMs and Decision Problems

- Suppose that M has exactly two halting states: Y and N.

TMs and Decision Problems

- Suppose that M has exactly two halting states: Y and N.
- Let $w \in \Sigma^*$ (input string) and denote:

$$f_M(w) = \begin{cases} 1 & \text{if } M \text{ terminates in } Y \text{ on input } w \\ 0 & \text{if } M \text{ terminates in } N \text{ on input } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

TMs and Decision Problems

- Suppose that M has exactly two halting states: Y and N.
- Let $w \in \Sigma^*$ (input string) and denote:

$$f_M(w) = \begin{cases} 1 & \text{if } M \text{ terminates in } Y \text{ on input } w \\ 0 & \text{if } M \text{ terminates in } N \text{ on input } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

- Let A be a decision problem and let M be a TM.

TMs and Decision Problems

- Suppose that M has exactly two halting states: Y and N.
- Let $w \in \Sigma^*$ (input string) and denote:

$$f_M(w) = \begin{cases} 1 & \text{if } M \text{ terminates in } Y \text{ on input } w \\ 0 & \text{if } M \text{ terminates in } N \text{ on input } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

- Let A be a decision problem and let M be a TM.
- If $f_M(w) = 1$ for all yes-instances w, and $f_M(w) = 0$ for all no-instances w, then M solves A.

TMs and Decision Problems

- Suppose that M has exactly two halting states: Y and N.
- Let $w \in \Sigma^*$ (input string) and denote:

$$f_M(w) = \begin{cases} 1 & \text{if } M \text{ terminates in } Y \text{ on input } w \\ 0 & \text{if } M \text{ terminates in } N \text{ on input } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

- Let A be a decision problem and let M be a TM.
- If $f_M(w) = 1$ for all yes-instances w, and $f_M(w) = 0$ for all no-instances w, then M solves A.
- E.g., for TSDP, if whenever

Input : Encoding of weighted graph and an value d

TMs and Decision Problems

- Suppose that M has exactly two halting states: Y and N.
- Let $w \in \Sigma^*$ (input string) and denote:

$$f_M(w) = \begin{cases} 1 & \text{if } M \text{ terminates in } Y \text{ on input } w \\ 0 & \text{if } M \text{ terminates in } N \text{ on input } w \\ \text{Undefined} & \text{Otherwise} \end{cases}.$$

- Let A be a decision problem and let M be a TM.
- If $f_M(w) = 1$ for all yes-instances w, and $f_M(w) = 0$ for all no-instances w, then M solves A.
- E.g., for TSDP, if whenever

$$\text{Input : Encoding of weighted graph and an value } d$$

$$\text{Outcome : } f_M(w) = 1 \text{ iff there's a } (\leq d) \text{ path covering all vertices.}$$

then M solves TSDP.

Using a solver for a decision problem to solve an optimization problem

Suppose that M solves TSDP, so $f_M(w) = 1$ if w codes a yes-instance of TSDP and $f_M(w) = 0$ otherwise.

Using a solver for a decision problem to solve an optimization problem

Suppose that M solves TSDP, so $f_M(w) = 1$ if w codes a yes-instance of TSDP and $f_M(w) = 0$ otherwise.

Here is a solver for the optimization problem TSP.

Using a solver for a decision problem to solve an optimization problem

Suppose that M solves TSDP, so $f_M(w) = 1$ if w codes a yes-instance of TSDP and $f_M(w) = 0$ otherwise.

Here is a solver for the optimization problem TSP.

Let (G, w) be an integer weighted graph.

Using a solver for a decision problem to solve an optimization problem

Suppose that M solves TSDP, so $f_M(w) = 1$ if w codes a yes-instance of TSDP and $f_M(w) = 0$ otherwise.

Here is a solver for the optimization problem TSP.

Let (G, w) be an integer weighted graph.

For $d = 0$ to $|G| \times$ max. edge weight
    if $f_M((G, w), d) = 1$ then return(d);

Using a solver for a decision problem to solve an optimization problem

Suppose that M solves TSDP, so $f_M(w) = 1$ if w codes a yes-instance of TSDP and $f_M(w) = 0$ otherwise.

Here is a solver for the optimization problem TSP.

Let (G, w) be an integer weighted graph.

For $d = 0$ to $|G|\times$ max. edge weight
    if $f_M((G, w), d) = 1$ then return(d);

- Consider the (unlikely) case that M terminates in p-time. Would the above code make a p-time algorithm for TSP?

Using a solver for a decision problem to solve an optimization problem

Suppose that M solves TSDP, so $f_M(w) = 1$ if w codes a yes-instance of TSDP and $f_M(w) = 0$ otherwise.

Here is a solver for the optimization problem TSP.

Let (G, w) be an integer weighted graph.

For $d = 0$ to $|G| \times$ max. edge weight
    if $f_M((G, w), d) = 1$ then return(d);

- Consider the (unlikely) case that M terminates in p-time. Would the above code make a p-time algorithm for TSP?
- Hint: How many bits are needed to encode a weighted graph?

Reduction

Let A, B be two decision problems. Let M be a deterministic TM.

Reduction

Let A, B be two decision problems. Let M be a deterministic TM. Suppose, for any encoding I of an instance of A that $f_M(I)$ is defined and is the encoding of some instance of B.

Reduction

Let A, B be two decision problems. Let M be a deterministic TM.
Suppose, for any encoding I of an instance of A that $f_M(I)$ is defined and is the encoding of some instance of B.
Suppose that

$$I \text{ is a yes-inst. of } A \iff f_M(I) \text{ is a yes-inst. of } B$$

Reduction

Let A, B be two decision problems. Let M be a deterministic TM. Suppose, for any encoding I of an instance of A that $f_M(I)$ is defined and is the encoding of some instance of B.
Suppose that

$$I \text{ is a yes-inst. of } A \iff f_M(I) \text{ is a yes-inst. of } B$$

Then A reduces to B.

Reduction

Let A, B be two decision problems. Let M be a deterministic TM. Suppose, for any encoding I of an instance of A that $f_M(I)$ is defined and is the encoding of some instance of B.
Suppose that

$$I \text{ is a yes-inst. of } A \iff f_M(I) \text{ is a yes-inst. of } B$$

Then A <u>reduces</u> to B.
If M runs in p-time then <u>A reduces to B in p-time</u> and we write

$$A \leq_p B$$

Reduction

Let A, B be two decision problems. Let M be a deterministic TM. Suppose, for any encoding I of an instance of A that $f_M(I)$ is defined and is the encoding of some instance of B.
Suppose that

$$I \text{ is a yes-inst. of } A \iff f_M(I) \text{ is a yes-inst. of } B$$

Then A reduces to B.
If M runs in p-time then A reduces to B in p-time and we write

$$A \leq_p B$$

Similarly, if $L, L' \subseteq \Sigma^*$ are languages and there exists p-time machine M such that

$$w \in \mathcal{L} \iff f_M(w) \in \mathcal{L}'$$

then

$$\mathcal{L} \leq_p \mathcal{L}'$$

Properties of $\leq_p$

Properties of $\leq_p$

Reflexive

$$A \leq_p A$$

Properties of $\leq_p$

Reflexive

$$A \leq_p A$$

Easy. Use the 'skip' machine.

Properties of $\leq_p$

Reflexive

$$A \leq_p A$$

Easy. Use the 'skip' machine.

Transitive

$$A \leq_p B \wedge B \leq_p C \Rightarrow A \leq_p C$$

Proof of transitivity

- Let M reduce A to B and let N reduce B to C. Then the composite TM $(M \circ N)$ reduces A to C.

Proof of transitivity

- Let M reduce A to B and let N reduce B to C. Then the composite TM $(M \circ N)$ reduces A to C.
- But how long does $(M \circ N)$ run?

Proof of transitivity

- Let M reduce A to B and let N reduce B to C. Then the composite TM $(M \circ N)$ reduces A to C.
- But how long does $(M \circ N)$ run?
- M runs in time $p(n)$ (some polynomial p) and N runs in time $q(n)$ (some polynomial q). So $(M \circ N)$ runs in time $p(n) + q(n)$ — still a polynomial.

Proof of transitivity

- Let M reduce A to B and let N reduce B to C. Then the composite TM $(M \circ N)$ reduces A to C.
- But how long does $(M \circ N)$ run?
- M runs in time $p(n)$ (some polynomial p) and N runs in time $q(n)$ (some polynomial q). So $(M \circ N)$ runs in time $p(n) + q(n)$ — still a polynomial.

WRONG

Run time

- Take an instance I of A of size n.

Run time

- Take an instance I of A of size n.
- M runs in time $p(n)$ and $f_M(I)$ is some instance of B.

Run time

- Take an instance I of A of size n.
- M runs in time $p(n)$ and $f_M(I)$ is some instance of B.
- $f_M(I)$ has a maximum size of $p(n)$.

Run time

- Take an instance I of A of size n.
- M runs in time $p(n)$ and $f_M(I)$ is some instance of B.
- $f_M(I)$ has a maximum size of $p(n)$.
- After running M we rewind the tape, this takes a maximum time of $p(n)$.

Run time

- Take an instance I of A of size n.
- M runs in time $p(n)$ and $f_M(I)$ is some instance of B.
- $f_M(I)$ has a maximum size of $p(n)$.
- After running M we rewind the tape, this takes a maximum time of $p(n)$.
- Now we run N on input size of at most $p(n)$. So N runs in time

$$q(p(n)).$$

Run time

- Take an instance I of A of size n.
- M runs in time $p(n)$ and $f_M(I)$ is some instance of B.
- $f_M(I)$ has a maximum size of $p(n)$.
- After running M we rewind the tape, this takes a maximum time of $p(n)$.
- Now we run N on input size of at most $p(n)$. So N runs in time

$$q(p(n)).$$

- This is still a polynomial!

Run time

- Take an instance I of A of size n.
- M runs in time $p(n)$ and $f_M(I)$ is some instance of B.
- $f_M(I)$ has a maximum size of $p(n)$.
- After running M we rewind the tape, this takes a maximum time of $p(n)$.
- Now we run N on input size of at most $p(n)$. So N runs in time

$$q(p(n)).$$

- This is still a polynomial!

  E.g., if $p(n) = n^2 + n$ and $q(n) = n^3 \log n$ then $q(p(n)) = (n^2 + n)^3 \log(n^2 + n) = O(n^6)$.

Run time

- Take an instance I of A of size n.
- M runs in time $p(n)$ and $f_M(I)$ is some instance of B.
- $f_M(I)$ has a maximum size of $p(n)$.
- After running M we rewind the tape, this takes a maximum time of $p(n)$.
- Now we run N on input size of at most $p(n)$. So N runs in time

$$q(p(n)).$$

- This is still a polynomial!

  E.g., if $p(n) = n^2 + n$ and $q(n) = n^3 \log n$ then $q(p(n)) = (n^2 + n)^3 \log(n^2 + n) = O(n^6)$.

- Therefore $M \circ N$ runs in p-time and we get $A \leq_p C$.

Hamiltonian Circuit Problem (HCP)

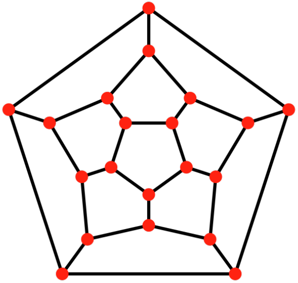
Instance: a graph G.

Yes-instance: if G has a Hamiltonian Circuit.
No-instance: otherwise.

Hamiltonian Circuit Problem (HCP)

Instance: a graph G.

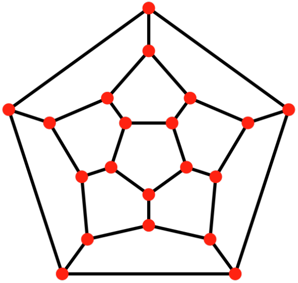
Yes-instance: if G has a Hamiltonian Circuit.
No-instance: otherwise.

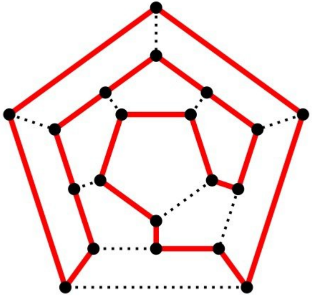


Hamiltonian Circuit Problem (HCP)

Instance: a graph G.

Yes-instance: if G has a Hamiltonian Circuit.
No-instance: otherwise.





Example: reducing HCP to TSDP

- Let $G = (V, E)$ be an instance of HCP.

Example: reducing HCP to TSDP

- Let $G = (V, E)$ be an instance of HCP.
- Let G' be a complete graph with same vertices, V, as G and weighting function defined by

$$w(v, v') = \begin{cases} 1 & \text{if } (v, v') \in E \\ 2 & \text{otherwise} \end{cases} .$$

Example: reducing HCP to TSDP

- Let $G = (V, E)$ be an instance of HCP.
- Let G' be a complete graph with same vertices, V, as G and weighting function defined by

$$w(v, v') = \begin{cases} 1 & \text{if } (v, v') \in E \\ 2 & \text{otherwise} \end{cases} .$$

The reduction maps $G = (V, E)$ to the instance $((V, w), |V|)$ of TSDP. Clearly, this reduction can be computed in time p-time.

Correctness of the reduction

- If G is a yes-instance of HCP then it has a Hamiltonian Circuit, say γ.

Correctness of the reduction

- If G is a yes-instance of HCP then it has a Hamiltonian Circuit, say γ.
- γ is a Hamiltonian Circuit of G' and the total weight of γ is $|V|$. Therefore $(G', |V|)$ is a yes-instance of TSDP.

Correctness of the reduction

- If G is a yes-instance of HCP then it has a Hamiltonian Circuit, say γ.
- γ is a Hamiltonian Circuit of G' and the total weight of γ is $|V|$. Therefore $(G', |V|)$ is a yes-instance of TSDP.
- Conversely, if $(G', |V|)$ is a yes-instance of TSDP then there must be a circuit ρ of G' of total weight not more than $|V|$, this can only happen if all nodes in the sequence are distinct and total weight equals $|V|$, hence ρ must be a Hamiltonian Circuit of G and so G is a yes-instance of HCP.

Conclusions of $HCP \leq_p TSDP$

Roughly:

Conclusions of $HCP \leq_p TSDP$

Roughly:

- A fast algorithm that solves TSDP could be turned into a fast algorithm to solve HCP.

Conclusions of $HCP \leq_p TSDP$

Roughly:

- A fast algorithm that solves TSDP could be turned into a fast algorithm to solve HCP.
- If there are no fast algorithms to solve HCP then there can be no fast algorithms for TSDP.

Lecture 3: Recap

Today we saw:

Lecture 3: Recap

Today we saw:

- How to craft (polynomial-time) reductions.

Today we saw:

- How to craft (polynomial-time) reductions.
- How optimization and decision problems differ, and how to go from one to the other.

Lecture 3: Recap

Today we saw:

- How to craft (polynomial-time) reductions.
- How optimization and decision problems differ, and how to go from one to the other.
- That not all (computable) problems are tractable.

Lecture 3: Recap

Today we saw:

- How to craft (polynomial-time) reductions.
- How optimization and decision problems differ, and how to go from one to the other.
- That not all (computable) problems are tractable.

Next week: Non-determinism and the $P \stackrel{?}{=} NP$ question.