Computational Complexity

Marianna Bonanome
City Tech

MAT 2440 - 2019

Computational Complexity - Hard, Harder, Hardest

Efficient Algorithms

Some polynomial time algorithms we learned about this semester:

1. The linear search procedure:

Computational Complexity - Hard, Harder, Hardest

Efficient Algorithms

Some polynomial time algorithms we learned about this semester:

1. The linear search procedure:$\Theta(n)$.
2. The bubble sort procedure:

Computational Complexity - Hard, Harder, Hardest

Efficient Algorithms

Some polynomial time algorithms we learned about this semester:

1. The linear search procedure: $\Theta(n)$.
2. The bubble sort procedure: $\Theta(n^2)$.
3. Matrix multiplication:

Computational Complexity - Hard, Harder, Hardest

Efficient Algorithms

Some polynomial time algorithms we learned about this semester:

1. The linear search procedure: $\Theta(n)$.
2. The bubble sort procedure: $\Theta(n^2)$.
3. Matrix multiplication: $\Theta(n^3)$.

Tractability

A problem that is solvable using an algorithm with polynomial (or better) worst-complexity is called **tractable**, because the expectation is that the algorithm will produce the solution to the problem for reasonably sized input in a relatively short time.

Computational Complexity - Hard, Harder, Hardest

Efficient Algorithms

Some polynomial time algorithms we learned about this semester:

1. The linear search procedure: $\Theta(n)$.
2. The bubble sort procedure: $\Theta(n^2)$.
3. Matrix multiplication: $\Theta(n^3)$.

Tractability

A problem that is solvable using an algorithm with polynomial (or better) worst-complexity is called **tractable**, because the expectation is that the algorithm will produce the solution to the problem for reasonably sized input in a relatively short time.

Problems that cannot be solved using an algorithm with worst-case polynomial time complexity are called **intractable**.

Solvability

Some problems exist for which it can be shown that no algorithm exists for solving them. Such problems are called **unsolvable**.

Solvability

Some problems exist for which it can be shown that no algorithm exists for solving them. Such problems are called **unsolvable**.

For example, we learned earlier in the semester that in 1936 Alan Turing showed that the **halting problem** is unsolvable.

Recall that the halting problem asks whether there is a procedure that takes as input 1) a computer program and 2) input the the program and determines whether the program will eventually stop when run with this input. (The proof is by contradiction!)

Some important classes of problems

P and NP

The class problems which are tractable is denoted by P.

Some important classes of problems

P and NP

The class problems which are tractable is denoted by P.

There are many problems for for which a solution, once found, can be recognized as correct in polynomial time - even though the solution itself might be hard to find (no poly-time algorithm to find the solution). The class of these problems is referred to as NP.

Some important classes of problems

P and NP

The class problems which are tractable is denoted by P.

There are many problems for for which a solution, once found, can be recognized as correct in polynomial time - even though the solution itself might be hard to find (no poly-time algorithm to find the solution). The class of these problems is referred to as NP.

A familiar NP problem

The factoring problem is in NP but outside of P because no known algorithm for a classical computer can solve it in only a polynomial number of steps - instead the number of steps increases exponentially as n increases. We will come back to this problem later!

Some important classes of problems

P and *NP*

The class problems which are tractable is denoted by P.

There are many problems for for which a solution, once found, can be recognized as correct in polynomial time - even though the solution itself might be hard to find (no poly-time algorithm to find the solution). The class of these problems is referred to as NP.

A familiar *NP* problem

The factoring problem is in NP but outside of P because no known algorithm for a classical computer can solve it in only a polynomial number of steps - instead the number of steps increases exponentially as n increases. We will come back to this problem later!

NP-complete

There is a class of problems with the property that if any of these problems can be solved by a an efficient algorithm, then all problems in the class can be solved by an efficient algorithm. They are in essence the "same" problem!!!

Examples of **NP-complete problems**

1. Given the dimensions of various boxes and want a way to pack them in your trunk.

Examples of **NP-complete problems**

1. Given the dimensions of various boxes and want a way to pack them in your trunk.
2. Given a map, color each country red, blue or green so that no two neighboring countries are the same.

Examples of **NP-complete problems**

1. Given the dimensions of various boxes and want a way to pack them in your trunk.
2. Given a map, color each country red, blue or green so that no two neighboring countries are the same.
3. Given a list of island connected by bridges and you want a tour which visits each island exactly once. If you want to find the shortest route, then this is known as the "Traveling Salesperson Problem."

Examples of **NP-complete problems**

1. Given the dimensions of various boxes and want a way to pack them in your trunk.
2. Given a map, color each country red, blue or green so that no two neighboring countries are the same.
3. Given a list of island connected by bridges and you want a tour which visits each island exactly once. If you want to find the shortest route, then this is known as the "Traveling Salesperson Problem."
4. Every known algorithm for these problems will take an amount of time that increases **exponentially** with the problem size.

Examples of **NP-complete problems**

1. Given the dimensions of various boxes and want a way to pack them in your trunk.
2. Given a map, color each country red, blue or green so that no two neighboring countries are the same.
3. Given a list of island connected by bridges and you want a tour which visits each island exactly once. If you want to find the shortest route, then this is known as the "Traveling Salesperson Problem."
4. Every known algorithm for these problems will take an amount of time that increases **exponentially** with the problem size.
5. These are all the "same" in that an efficient algorithm for solving one of them will imply an efficient algorithm for solving all of them.

$P = NP$?

The million dollar question (literally)

An efficient algorithm for an NP-**complete** problem would mean that computer scientists' present picture of the classes P, NP and NP-**complete** was utterly wrong!!! It would mean that $P = NP$!

$P = NP$?

The million dollar question (**literally**)

An efficient algorithm for an NP-**complete** problem would mean that computer scientists' present picture of the classes P, NP and NP-**complete** was utterly wrong!!! It would mean that $P = NP$!

Does such an algorithm exist? This question carries a $1,000,000 reward from the Clay Math Institute in Cambridge, Mass.

$P = NP$?

The million dollar question (literally)

An efficient algorithm for an NP-**complete** problem would mean that computer scientists' present picture of the classes P, NP and NP-**complete** was utterly wrong!!! It would mean that $P = NP$!

Does such an algorithm exist? This question carries a $1,000,000 reward from the Clay Math Institute in Cambridge, Mass.

If we grant that $P \neq NP$, our only hope is to broaden what we mean by "computer."

$P = NP$?

The million dollar question (**literally**)

An efficient algorithm for an NP-**complete** problem would mean that computer scientists' present picture of the classes P, NP and NP-**complete** was utterly wrong!!! It would mean that $P = NP$!

Does such an algorithm exist? This question carries a $1,000,000 reward from the Clay Math Institute in Cambridge, Mass.

If we grant that $P \neq NP$, our only hope is to broaden what we mean by "computer."
NOTE: The factoring problem is neither known nor believed to be NP-**complete**.

NP-complete problems, practically speaking

In January 2018, I attended the Joing Mathematics Meeting in San Diego and went to a talk by William Cook of the University of Waterloo call "Information, Computation, Optimization: Connecting the Dots in the Traveling Salesman Problem."

NP-complete problems, practically speaking

In January 2018, I attended the Joing Mathematics Meeting in San Diego and went to a talk by William Cook of the University of Waterloo call "Information, Computation, Optimization: Connecting the Dots in the Traveling Salesman Problem."

According to Prof. Cook: "The popular interpretation is that we simply cannot solve realistic examples. But this skips over nearly 70 years of intense mathematical study!" He used linear programming methods to show that a certain tour of 49,603 historic sites in the U.S. is shortest possible.

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Moore's Law

Computer power will double for constant cost roughly every two years.

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Moore's Law

Computer power will double for constant cost roughly every two years.

Prediction

This dream will come to an end during this decade.

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Moore's Law

Computer power will double for constant cost roughly every two years.

Prediction

This dream will come to an end during this decade.

Quantum effects are beginning to interfere with electronic devices as they are made smaller and smaller.

Solution

Move to a different paradigm!

Solution

Move to a different paradigm!

Can we use the counterintuitive laws of quantum mechanics to our advantage?

Solution

Move to a different paradigm!

Can we use the counterintuitive laws of quantum mechanics to our advantage?

Yes! In 1982 Richard Feynman and Paul Benioff independently observed that a quantum system can perform a computation.

Solution

Move to a different paradigm!

Can we use the counterintuitive laws of quantum mechanics to our advantage?

Yes! In 1982 Richard Feynman and Paul Benioff independently observed that a quantum system can perform a computation.

In 1985 David Deutsch defined quantum Turing machines, a theoretical model for quantum computing.

References:

- Preskill's podcast and notes: https://quantumfrontiers.com/author/preskill/
- A timeline on quantum computation: https://en.wikipedia.org/wiki/Timeline_of_quantum_computing
- An article, "The Limits of Quantum Computers" https://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf
- The "bible" for learning quantum computation: http://csis.pace.edu/ctappert/cs837-18spring/QC-textbook.pdf
- Another perspective: https://www.worldscientific.com/worldscibooks/10.1142/3808