Computational Complexity: Classes

AiC FME, UPC

Fall 2021

Classifying decidable problems

- Decidable problems, admit algorithmic solutions.
- The algorithms solving such problems can use more or less amount of resources.
- To measure the performance of an algorithm we use two measures time and space.
- As usual we take a worst case analysis on inputs with the same size.

Classifying decidable problems

- Decidable problems, admit algorithmic solutions.
- The algorithms solving such problems can use more or less amount of resources.
- To measure the performance of an algorithm we use two measures time and space.
- As usual we take a worst case analysis on inputs with the same size.
- For a TM, time correspond to the length of the computation and space to the portion of the tape accessed during the computation.

Classifying decidable problems

- Decidable problems, admit algorithmic solutions.
- The algorithms solving such problems can use more or less amount of resources.
- To measure the performance of an algorithm we use two measures time and space.
- As usual we take a worst case analysis on inputs with the same size.
- For a TM, time correspond to the length of the computation and space to the portion of the tape accessed during the computation.
- Observe that sometimes the space used might be smaller than the input.

Complexity classes: definitions

Let $f : \mathbb{N} \to \mathbb{N}$,

- The class $\mathrm{TIME}(f(n))$ is the class of decision problems for which an algorithm exists that solves instances of size n in time $O(f(n))$.
- The class $\mathrm{SPACE}(f(n))$ is the class of decision problems for which an algorithm exists that solves instances of size n using space $O(f(n))$.

Some complexity classes

- P $= \cup_k TIME(n^k) = TIME(poly(n))$
- EXP $= TIME(2^{poly(n)})$
- 2EXP $= TIME(2^{2^{poly(n)}})$
- L = LOGSPACE $= SPACE(\lg n)$
- PSPACE $= SPACE(poly(n))$
- EXPSPACE $= SPACE(2^{poly(n)})$

Some complexity classes

- P $= \cup_k TIME(n^k) = TIME(poly(n))$
- EXP $= TIME(2^{poly(n)})$
- 2EXP $= TIME(2^{2^{poly(n)}})$
- L = LOGSPACE $= SPACE(\lg n)$
- PSPACE $= SPACE(poly(n))$
- EXPSPACE $= SPACE(2^{poly(n)})$

$$\mathrm{L} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPSPACE}$$

Some complexity classes

- P $= \cup_k TIME(n^k) = TIME(poly(n))$
- EXP $= TIME(2^{poly(n)})$
- 2EXP $= TIME(2^{2^{poly(n)}})$
- L = LOGSPACE $= SPACE(\lg n)$
- PSPACE $= SPACE(poly(n))$
- EXPSPACE $= SPACE(2^{poly(n)})$

$$\mathrm{L} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPSPACE}$$

$$\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{EXP} \qquad \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

Nondeterministic TM

A non deterministic Turing machine (NTM) M is a tuple $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_F)$, where

- Q is a finite set of states,
- Σ is the input alphabet.
- Γ is the tape alphabet, $\Gamma = \Sigma \cup \{\mathtt{b}, \blacktriangleright\}$, with $\mathtt{b}, \blacktriangleright \notin \Sigma$.
- $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\mathtt{l}, \mathtt{r}, \mathtt{n}\}$ is the transition relation.
- q_0 is the initial state.
- q_F is the final or accepting state.

Non deterministic Turing machines: Recognizing languages

- Let $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_F)$ be a TM and $x \in \Sigma^*$
- The computation of M with input x goes as follows:
  - Initially: the state is q_0; the tape has $\blacktriangleright x$ and all remaining cells in the tape hold a $\mathtt{b}$; the head has access to the first symbol of x.
  - While there is a transition in δ for the combination state, symbol accessed by the head, one of such transitions is applied.

Non deterministic Turing machines: Recognizing languages

- Let $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_{\mathrm{F}})$ be a TM and $x \in \Sigma^*$
- The computation of M with input x goes as follows:
  - Initially: the state is q_0; the tape has $\blacktriangleright x$ and all remaining cells in the tape hold a $\mathtt{b}$; the head has access to the first symbol of x.
  - While there is a transition in δ for the combination state, symbol accessed by the head, one of such transitions is applied.
- Assuming that Q and Γ are disjoint, the word $\blacktriangleright \alpha q \beta$ is a configuration in which $\blacktriangleright \alpha\beta$ are the tape contents ($\mathtt{b}$ outside), q is a state and the head is accessing the tape cell holding the first symbol in β.

Non deterministic Turing machines: Recognizing languages

- Let $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_{\mathrm{F}})$ be a TM and $x \in \Sigma^*$
- The computation of M with input x goes as follows:
  - Initially: the state is q_0; the tape has $\blacktriangleright x$ and all remaining cells in the tape hold a $\mathtt{b}$; the head has access to the first symbol of x.
  - While there is a transition in δ for the combination state, symbol accessed by the head, one of such transitions is applied.
- Assuming that Q and Γ are disjoint, the word $\blacktriangleright \alpha q \beta$ is a configuration in which $\blacktriangleright \alpha\beta$ are the tape contents ($\mathtt{b}$ outside), q is a state and the head is accessing the tape cell holding the first symbol in β.
- The computation of M on x is a rooted tree of configurations, the root is $\blacktriangleright q_0 x$, so that we can pass from father to son selecting a component in Δ.

Non deterministic Turing machines: Recognizing languages

- The computation of a TM on input x is a tree of configurations.
- If the tree of configurations has a leaf, we say that M halts on input x, we note this as $M(x)\downarrow$, otherwise, the computation diverges or does not halt, $M(x)\uparrow$.

Non deterministic Turing machines: Recognizing languages

- The computation of a TM on input x is a tree of configurations.
- If the tree of configurations has a leaf, we say that M halts on input x, we note this as $M(x)\downarrow$, otherwise, the computation diverges or does not halt, $M(x)\uparrow$.
- M accepts x if $M(x)\downarrow$ and there is a leaf in the computation with a configuration with state q_{F}.

Non deterministic Turing machines: Recognizing languages

- The computation of a TM on input x is a tree of configurations.
- If the tree of configurations has a leaf, we say that M halts on input x, we note this as $M(x)\downarrow$, otherwise, the computation diverges or does not halt, $M(x)\uparrow$.
- M accepts x if $M(x)\downarrow$ and there is a leaf in the computation with a configuration with state q_{F}.
- $L(M) \subseteq \Sigma^*$ is the set of words that M accepts.

Non deterministic Turing machines: Recognizing languages

Theorem

A language $L \subseteq \Sigma^*$ is recognizable iff there is a NTM M with $L = L(M)$.

Non deterministic Turing machines: Deciding languages

- We need a stopping criteria to define a non deterministic decider.

Non deterministic Turing machines: Deciding languages

- We need a stopping criteria to define a non deterministic decider.
- A NTM M is a decider if, for each input x, the configuration tree is finite.

Non deterministic Turing machines: Deciding languages

- We need a stopping criteria to define a non deterministic decider.
- A NTM M is a decider if, for each input x, the configuration tree is finite.

Theorem

A language $L \subseteq \Sigma^*$ is decidable iff there is a decider NTM M such that $L = L(M)$.

Non deterministic Turing machines: Recognizing languages

- In a decider, the computation tree is always finite.

Non deterministic Turing machines: Recognizing languages

- In a decider, the computation tree is always finite.
- The number of configurations in the longest path from root to leaves is the computation time.
- In a similar way the space used is the required for the worst computation path.

Complexity classes: definitions

Let $f : \mathbb{N} \to \mathbb{N}$,

- The class NTIME($f(n)$) is the class of decision problems that a non deterministic TM recognizes in time $O(f(n))$.
- The class NSPACE($f(n)$) is the class of decision problems for which a non deterministic TM solves instances of size n using space $O(f(n))$.

Some complexity classes

- NP $= \text{NTIME}(poly(n))$
- NEXP $= \text{NTIME}(2^{poly(n)})$
- NL $= \text{NSPACE}(\lg n)$
- NPSPACE $= \text{NSPACE}(poly(n))$

Some complexity classes

- NP $= \text{NTIME}(poly(n))$
- NEXP $= \text{NTIME}(2^{poly(n)})$
- NL $= \text{NSPACE}(\lg n)$
- NPSPACE $= \text{NSPACE}(poly(n))$

$$\text{L} \subseteq \text{NL} \subseteq \text{P}$$

Some complexity classes

- NP $=$ NTIME($poly(n)$)
- NEXP $=$ NTIME($2^{poly(n)}$)
- NL $=$ NSPACE($\lg n$)
- NPSPACE $=$ NSPACE($poly(n)$)

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P}$$

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXP}$$

Logic and NP

- A verifier for a language L is an algorithm $\mathcal{A}$, where $L = \{x \mid \mathcal{A} \; accepts\langle x, c\rangle$ for some word $c\}$.

Logic and NP

- A verifier for a language L is an algorithm $\mathcal{A}$, where $L = \{x \mid \mathcal{A}\ \mathit{accepts} \langle x, c\rangle$ for some word $c\}$.
- A polynomial time verifier runs in polynomial time in the length of x.
- A language L is polynomially verifiable if it has a polynomial time verifier.
- A verifier uses additional information, represented by the word c, to verify that a string x is a member of L. This information is called a certificate, or proof, of membership in L.
- A polynomial verifier can access only polynomial space. Therefore, we can assume that $|c|$ is polynomial in $|x|$.

Decision Problems in NP: Examples

Theorem

$L \in NP$ iff $L = \{x \mid \exists y\, R(x, y)\}$ where $|y| = poly(|x|)$ and R can be decided in polynomial time.

Decision Problems in NP: Examples

Theorem

$L \in NP$ iff $L = \{x \mid \exists y\, R(x, y)\}$ where $|y| = poly(|x|)$ and R can be decided in polynomial time.

NP is the class of polynomially verifiable languages.

Decision Problems in NP: Examples

- COMPOSITE: Given an integer n, is n composite?

Decision Problems in NP: Examples

- COMPOSITE: Given an integer n, is n composite?
  - $\textsc{Composite} = \{n \mid \exists n_1, n_2 \; n = n_1 \times n_2\}$
  - The certificate has polynomial size, and the verifier runs in polynomial time.
  - So, $\textsc{Composite} \in \mathsf{NP}$.

Decision Problems in NP: Examples

- COMPOSITE: Given an integer n, is n composite?
    - $\textsc{Composite} = \{n \mid \exists n_1, n_2\ n = n_1 \times n_2\}$
    - The certificate has polynomial size, and the verifier runs in polynomial time.
    - So, $\textsc{Composite} \in \mathrm{NP}$.
- SUBSET-SUM: Given a set $S = \{x_1, \ldots, x_n\}$ of $\mathbb{Z}^+$ and $t \in \mathbb{Z}$, exists $S' \subseteq S$ with $\sum_{x_j \in S'} x_j = t$?

Decision Problems in NP: Examples

- COMPOSITE: Given an integer n, is n composite?
  - $\textsc{Composite} = \{n \mid \exists n_1, n_2 \; n = n_1 \times n_2\}$
  - The certificate has polynomial size, and the verifier runs in polynomial time.
  - So, $\textsc{Composite} \in \mathrm{NP}$.
- SUBSET-SUM: Given a set $S = \{x_1, \ldots, x_n\}$ of $\mathbb{Z}^+$ and $t \in \mathbb{Z}$, exists $S' \subseteq S$ with $\sum_{x_j \in S'} x_j = t$?
  - The certificate is a subset of S, it can be represented by a boolean vector, so it has polynomial size. The verifier runs in polynomial time.
  - So, $\textsc{Subset-Sum} \in \mathrm{NP}$.

Decision Problems in NP: Examples

- COMPOSITE: Given an integer n, is n composite?
  - $\textsc{Composite} = \{n \mid \exists n_1, n_2 \; n = n_1 \times n_2\}$
  - The certificate has polynomial size, and the verifier runs in polynomial time.
  - So, $\textsc{Composite} \in \text{NP}$.
- SUBSET-SUM: Given a set $S = \{x_1, \ldots, x_n\}$ of $\mathbb{Z}^+$ and $t \in \mathbb{Z}$, exists $S' \subseteq S$ with $\sum_{x_j \in S'} x_j = t$?
  - The certificate is a subset of S, it can be represented by a boolean vector, so it has polynomial size. The verifier runs in polynomial time.
  - So, $\textsc{Subset-Sum} \in \text{NP}$.
- HAMILTONIAN CYCLE: Given a graph G, is G Hamiltonian?

Decision Problems in NP: Examples

- COMPOSITE: Given an integer n, is n composite?
  - $\textsc{Composite} = \{n \mid \exists n_1, n_2\ n = n_1 \times n_2\}$
  - The certificate has polynomial size, and the verifier runs in polynomial time.
  - So, $\textsc{Composite} \in \mathrm{NP}$.
- SUBSET-SUM: Given a set $S = \{x_1, \ldots, x_n\}$ of $\mathbb{Z}^+$ and $t \in \mathbb{Z}$, exists $S' \subseteq S$ with $\sum_{x_j \in S'} x_j = t$?
  - The certificate is a subset of S, it can be represented by a boolean vector, so it has polynomial size. The verifier runs in polynomial time.
  - So, $\textsc{Subset-Sum} \in \mathrm{NP}$.
- HAMILTONIAN CYCLE: Given a graph G, is G Hamiltonian?
  - The certificate is a permutation of $V(G)$, it can be represented by a vector, so it has polynomial size. The verifier also runs in polynomial time.
  - So, $\textsc{Hamiltonian Cycle} \in \mathrm{NP}$.

Classes of complements

- Let $\mathcal{C}$ be a class of decision problems. co-$\mathcal{C}$ is formed by the complements of languages in $\mathcal{C}$, i.e., $L \in$ co-$\mathcal{C}$ iff $\overline{L} \in \mathcal{C}$.

Classes of complements

- Let $\mathcal{C}$ be a class of decision problems. co-$\mathcal{C}$ is formed by the complements of languages in $\mathcal{C}$, i.e., $L \in$ co-$\mathcal{C}$ iff $\overline{L} \in \mathcal{C}$.
- Classes defined by TMs are closed under complement.

$$\mathrm{P} = \text{co-P} \qquad \mathrm{EXP} = \text{co-EXP} \qquad \mathrm{PSPACE} = \text{co-PSPACE}$$

Classes of complements

- Let $\mathcal{C}$ be a class of decision problems. co-$\mathcal{C}$ is formed by the complements of languages in $\mathcal{C}$, i.e., $L \in$ co-$\mathcal{C}$ iff $\overline{L} \in \mathcal{C}$.
- Classes defined by TMs are closed under complement.

$$\text{P} = \text{co-P} \qquad \text{EXP} = \text{co-EXP} \qquad \text{PSPACE} = \text{co-PSPACE}$$

- Not so clear for classed defined by NTMs.
  co-NP $= \{x \mid \forall y R(x, y)\}$ where $|y| = poly(|x|)$ and R can be decided in polynomial time.
- It seems different to asses that a property holds for a leaf in a tree than on all their leaves.

Decision Problems in co-NP: Examples

- PRIME: Given an integer n, is n composite?
  - $\textsc{Prime} = \{n \mid \forall a \leq n \; n \mod a \neq 0\}$.
  - PRIME is the complement of COMPOSITE, we have $\textsc{Prime} \in$ co-NP

Decision Problems in co-NP: Examples

- PRIME: Given an integer n, is n composite?
  - $\textsc{Prime} = \{n \mid \forall a \leq n \; n \mod a \neq 0\}$.
  - PRIME is the complement of COMPOSITE, we have PRIME $\in$ co-NP
  - PRIME $\in NP$ as it can be verified in poly time (Pratt 1975). So in NP $\cap$ co-NP

Decision Problems in co-NP: Examples

- PRIME: Given an integer n, is n composite?
  - $\textsc{Prime} = \{n \mid \forall a \leq n \; n \mod a \neq 0\}$.
  - PRIME is the complement of COMPOSITE, we have PRIME $\in$ co-NP
  - PRIME $\in$ *NP* as it can be verified in poly time (Pratt 1975). So in NP $\cap$ co-NP
  - A polynomial time algorithm $O(\lg n^6)$ was devised by Agrawal, Kayal and Saxena in 2002.

Decision Problems in NP: Examples

- FACTORIZATION: Given an integer n, are there primes $p_1, \ldots, p_k$ s.t. $x = p_1 \times \ldots \times p_k$.

Decision Problems in NP: Examples

- FACTORIZATION: Given an integer n, are there primes $p_1, \ldots, p_k$ s.t. $x = p_1 \times \ldots \times p_k$.
  - $p_1, \ldots, p_k$ is the certificate.
  - k and all the possible factors p are $\leq n$.
  - The verifier, on input n and $p_1, \ldots, p_k$, computes $p_1 \times \ldots \times p_k$, checks if the result is n. If so, checks that all the values are prime numbers.
  - This takes time $O(k \lg n^2 + k \lg n^6)$.
  - So, FACTORIZATION $\in$ NP.

Open questions

- Is $\mathrm{P} = \mathrm{NP}$?
- $\mathrm{P} \neq \mathrm{EXP}$, but is $\mathrm{NP} = \mathrm{EXP}$?
- $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$, but is $\mathrm{P} = \mathrm{NP} \cap \text{co-NP}$?
- Is $\mathrm{L} = \mathrm{NL}$?

Efficient algorithms?

Feasible problem there is an algorithm to find a solution efficiently.

Efficient algorithms?

Feasible problem there is an algorithm to find a solution efficiently.

The Determinant Problem: Given an $n \times n$ matrix $M = (m_{ij})$, compute

$$\det(M) = \sum_{\pi \in S_n} (-1)^{\pi} \prod_{i=1}^{n} m_{i,\pi(i)},$$

where $(-1)^{\pi} = \begin{cases} -1 & \text{if } \pi \text{has odd parity,} \\ +1 & \text{if } \pi \text{ has even parity.} \end{cases}$

Efficient algorithms?

Feasible problem there is an algorithm to find a solution efficiently.

The Determinant Problem: Given an $n \times n$ matrix $M = (m_{ij})$, compute

$$\det(M) = \sum_{\pi \in S_n} (-1)^{\pi} \prod_{i=1}^{n} m_{i,\pi(i)},$$

where $(-1)^{\pi} = \begin{cases} -1 & \text{if } \pi \text{has odd parity,} \\ +1 & \text{if } \pi \text{ has even parity.} \end{cases}$

Can be solved in $O(n^3)$, using the LU decomposition.

Efficient algorithms?

The Permanent Problem: Given an $n \times n$ matrix $M = (m_{ij})$, compute

$$\mathrm{perm}(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} m_{i,\pi(i)}.$$

$$\mathrm{Perm} \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei + bfg + cdh + ceg + bdi + afh$$

Probably exponential Valiant 1979, best complexity known, $O(n2^n)$. Glynn 2010

Efficient algorithms?

The Permanent Problem: Given an $n \times n$ matrix $M = (m_{ij})$, compute

$$\mathrm{perm}(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} m_{i,\pi(i)}.$$

$$\mathrm{Perm} \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei + bfg + cdh + ceg + bdi + afh$$

Probably exponential Valiant 1979, best complexity known, $O(n2^n)$. Glynn 2010

Main difference $\det(M_1 \times M_2) = \det(M_1) \times \det(M_2)$ but $\mathrm{perm}(M_1 \times M_2) \neq \mathrm{perm}(M_1) \times \mathrm{perm}(M_2)$

Gödel's letter to von Neumann (1954)

Gödel considered the Truncated Entscheidungsproblem: *given any statement in first order-logic, decide if there is a proof of the statement with finite length of at most m lines*, where m could be any large but finite number.

Gödel's letter to von Neumann (1954)

Gödel considered the Truncated Entscheidungsproblem: *given any statement in first order-logic, decide if there is a proof of the statement with finite length of at most m lines*, where m could be any large but finite number.

For instance, decide if there is a proof with at most $m = 10^{10}$ lines to $\exists x, y, z, n \in \mathbb{N} - \{0\} \,:\, (n \geq 3) \wedge (x^n + y^n = z^n)$.

Gödel's letter to von Neumann (1954)

Gödel considered the Truncated Entscheidungsproblem: *given any statement in first order-logic, decide if there is a proof of the statement with finite length of at most m lines*, where m could be any large but finite number.

For instance, decide if there is a proof with at most $m = 10^{10}$ lines to $\exists x, y, z, n \in \mathbb{N} - \{0\} \ : (n \geq 3) \wedge (x^n + y^n = z^n)$.

The Truncated Entscheidungsproblem is decidable!

Gödel's letter to von Neumann (1954)

Gödel considered the Truncated Entscheidungsproblem: *given any statement in first order-logic, decide if there is a proof of the statement with finite length of at most m lines*, where m could be any large but finite number.

For instance, decide if there is a proof with at most $m = 10^{10}$ lines to $\exists x, y, z, n \in \mathbb{N} - \{0\} \ : (n \geq 3) \wedge (x^n + y^n = z^n)$.

The Truncated Entscheidungsproblem is decidable!

In his letter Gödel asked if there is a $O(m)$ or $O(m^2)$ algorithm to decide it.

Efficient algorithms?

- In computational complexity efficient algorithm is synonym of polynomial time.
- In practice, it is synonym of a low degree polynomial time.
- Nevertheless, we some times use efficient as synonym of best known cost.

Efficient algorithms?

- In computational complexity efficient algorithm is synonym of polynomial time.
- In practice, it is synonym of a low degree polynomial time.
- Nevertheless, we some times use efficient as synonym of best known cost.

- How to differentiate strict exponential time from polynomial time?
- We cannot do that in most of the cases, we relate such answers to some of the open complexity questions.

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

- Function problems
  Given x, compute y such that $R(x, y)$ holds.

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

- Function problems
  Given x, compute y such that $R(x, y)$ holds.
  Classes: FP, FNP, FEXP, ...

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

- Function problems
  Given x, compute y such that $R(x, y)$ holds.
  Classes: FP, FNP, FEXP, ...
- Optimization problems
  Given x, compute y such that $R(x, y)$ holds and $m(x, y)$ is maximum/minimum.

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

- Function problems
  Given x, compute y such that $R(x, y)$ holds.
  Classes: FP, FNP, FEXP, ...
- Optimization problems
  Given x, compute y such that $R(x, y)$ holds and $m(x, y)$ is maximum/minimum.
  Classes: PO, NPO, EXPO, ...

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

- Function problems
  Given x, compute y such that $R(x, y)$ holds.
  Classes: FP, FNP, FEXP, ...
- Optimization problems
  Given x, compute y such that $R(x, y)$ holds and $m(x, y)$ is maximum/minimum.
  Classes: PO, NPO, EXPO, ...
- Counting problems
  Given x, how many y are there such that $R(x, y)$?

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

- Function problems
  Given x, compute y such that $R(x, y)$ holds.
  Classes: FP, FNP, FEXP, ...
- Optimization problems
  Given x, compute y such that $R(x, y)$ holds and $m(x, y)$ is maximum/minimum.
  Classes: PO, NPO, EXPO, ...
- Counting problems
  Given x, how many y are there such that $R(x, y)$?
  Classes: #P

Function Problems: Examples

- Reachability: Given a direct graph G and two vertices u and v, find a path a path from u to v, if one exists.
- Eulerian cycle: Given a graph G, find an Eulerian cycle that traverses all the edges exactly once, if such a cycle exists.
- Factoring: Given an integer n, find its prime factors.
- Subset Sum: Given a sequence of positive integers $S = \{a_1, \ldots, a_n\}$ and $t \in \mathbb{Z}$, obtain $I \subseteq \{1, \ldots, n\}$ s.t. $\sum_{i \in I} a_i = t$.

Optimization Problems: Examples

- Max Cut: Given a graph G, find a partition a partition V into V_1, V_2, such that it maximizes the number of crossing edges between V_1 and V_2.
- Min Cut: Given a digraph G, find a partition V into V_1, V_2, such that it minimizes the number of edges going from V_1 to V_2.
- Shortest path: Given a connected and edge weighted digraph G and $s, t \in V(G)$, find a path between s and t minimizing the sum of the weights in the path.

Counting problems

- How many perfect matchings are there for a given bipartite graph?
- How many graph colorings using k colors are there for a particular graph G?
- What is the value of the permanent of a given matrix whose entries are 0 or 1?
- How many different variable assignments will satisfy a given general boolean formula?

More complexity classes?

Sure, plenty!
Look into the Complexity zoo.
Around 546 the last time I've had a look!