Computational complexity and combinatorial optimization

Jérôme Lang
IRIT – CNRS and Université Paul Sabatier
31062 Toulouse Cedex, France
lang@irit.fr
Plan of the course

1. Problems: decision, search and optimization.
2. Complexity classes, reductions, and completeness.
3. Above NP and coNP
4. Complexity classes for function and optimization problems
5. Approximation
6. Compact representation and compilation

Parts of this course are based on the following books:


and on the following paper:

1. Problems: decision, search and optimization

Problems: decision, search and optimization

Computer programming: designing algorithms for the *solution of problems*.

Which problems?

- decision problems
- search (or function) problems
- optimization problems
1. Problems: decision, search and optimization

Decision problems

A decision problem is a pair \( P = \langle I_P, Y_P \rangle \) where

- \( I_P \) set of problem instances
- \( Y_P \subseteq I_P \) set of positive instances

\[ N_P = I_P \setminus Y_P \text{ set of negative instances} \]

A decision problem is usually identified with the language \( Y_P \) of positive instances.

Algorithm for a decision problem:

A decision problem \( P \) is solved by an algorithm \( A \) if \( A \) halts for every instance \( x \in I_P \), and returns YES if and only if \( x \in Y_P \). We also say that the set (or the language) \( Y_P \) is recognized by \( A \).
1. Problems: decision, search and optimization

**Decision problems: examples**

SATISFIABILITY (or SAT):

**Instance** a CNF formula $\varphi$ on a set $V$ of Boolean variables;

**Question** is $\varphi$ satisfiable, i.e., does there exist a truth assignment $f : V \rightarrow \{\text{TRUE, FALSE}\}$ which satisfies $\varphi$?

$I_\varphi = \text{set of all well-formed CNF formulas on } V$;

$Y_\varphi = \text{set of all satisfiable CNF formulas on } V$.

GRAPH COLORING:

**Instance** a graph $G = \langle V, E \rangle$ and a positive integer $K$;

**Question** does there exist an assignment $f : V \rightarrow \{1, \ldots, K\}$ such that for each $(i, j) \in E$, $f(i) \neq f(j)$?

PRIME:

**Instance** an integer $k$

**Question** is $k$ a prime number?
1. Problems: decision, search and optimization

**MAXSAT**($K$):

**Instance** a CNF formula $\varphi = C_1 \land \ldots \land C_m$ on a set $V$ of Boolean variables, and an integer $K$;

**Question** does there exist a truth assignment $f : V \rightarrow \{\text{TRUE, FALSE}\}$ which satisfies at least $K$ clauses of $\varphi$?

**MAXIMUM SATISFYING TRUTH ASSIGNMENT:**

**Instance** a CNF formula $\varphi = C_1 \land \ldots \land C_m$ on a set $V$ of Boolean variables, and a truth assignment $f : V \rightarrow \{\text{TRUE, FALSE}\}$.

**Question** does $f$ satisfies a maximum number of clauses of $\varphi$?
1. Problems: decision, search and optimization

Search (or function) problems

Search problem: \( \mathcal{P} = \langle I_\mathcal{P}, S_\mathcal{P}, R \rangle \) where

- \( I_\mathcal{P} \) set of problem instances
- \( S_\mathcal{P} \) set of problem solutions
- \( R \subseteq I_\mathcal{P} \times S_\mathcal{P} \quad [R(x,s) \text{ means that } s \text{ is a solution for } x] \)

**FSAT:**

**Instance** a CNF formula \( \varphi \) on a set \( V \) of Boolean variables;

**Solution** a truth assignment \( f : V \rightarrow \{ \text{TRUE,FALSE} \} \) which satisfies \( \varphi \) if there exists one, or NO otherwise.

**FTSP:**

**Instance** a complete valued graph \( G = \langle V, D \rangle \) with \( V = \{ v_1, \ldots, v_n \} \) and

\[ D(v,v') = D(v',v) \in \mathbb{N} \text{ (distance between } v \text{ and } v') \]

**Solution** a Hamiltonian path of minimum length of \( G \).
Optimization problems

Optimization problem: $\mathcal{P} = \langle I_\mathcal{P}, SOL_\mathcal{P}, m_\mathcal{P}, \text{goal}_\mathcal{P} \rangle$ where

- $I_\mathcal{P}$ set of problem instances
- $SOL_\mathcal{P}$ is a function that associates to any input instance $x \in I_\mathcal{P}$ the set of feasible solutions of $x$;
- $m_\mathcal{P}$ is the measure function, defined for pairs $(x, s)$ such that $x \in I_\mathcal{P}$ and $s \in SOL_\mathcal{P}(x)$. For every such pair $(x, s)$, $m_\mathcal{P}(x, s)$ provides a positive integer which is the value of the feasible solution $s$.
- $\text{goal}_\mathcal{P} \in \{\text{MIN}, \text{MAX}\}$ specifies whether $\mathcal{P}$ is a minimization of a maximization problem.

Given $x \in I_\mathcal{P}$, we denote by $SOL^*_\mathcal{P}(x)$ the set of optimal solutions of $x$, that is, the set of solutions $s$ whose value $m_\mathcal{P}(s)$ is minimal (if $\text{goal} = \text{MIN}$) or maximal (if $\text{goal} = \text{MAX}$).

The value of any optimal solution $s^* \in SOL^*_\mathcal{P}(x)$ is denoted as $m^*_\mathcal{P}(x)$. 
1. Problems: decision, search and optimization

Optimization problems: examples

**MAXSAT**:

**Instance** a CNF formula $\varphi = C_1 \land \ldots \land C_m$ on a set $V$ of Boolean variables

**Solution** a truth assignment $f : V \rightarrow \{\text{TRUE, FALSE}\}$

**Measure** number of clauses $C_i$ such that $f$ satisfies $C_i$

**Goal** maximize

**MINIMUM GRAPH COLORING**:

**Instance** a graph $G = \langle V, E \rangle$;

**Solution** an assignment $f : V \rightarrow \{1, \ldots, K\}$ such that for each $(i, j) \in E$, $f(i) \neq f(j)$

**Measure** $K$

**Goal** minimize
1. Problems: decision, search and optimization

Optimization problems: examples (continued)

MIN-TSP:

**Instance**  a complete valued graph \( G = \langle V, D \rangle \)

**Solution**  a permutation \( \sigma \) of \( V \)

**Measure**  \( \sum_{k=1}^{n-1} D(v_{\sigma(i)}, v_{\sigma(i+1)}) + D(v_{\sigma(n)}, v_{\sigma(1)}) \)

**Goal**  minimize

MAXIMUM KNAPSACK:

**Instance**  a set \( X = \{x_1, \ldots, x_n\} \) of objects; for each object, a volume \( a_i \) and a value \( p_i \); a maximum volume \( b \).

**Solution**  a subset \( Y \subseteq X \) such that \( \sum_{x_i \in Y} a_i \leq b \);

**Measure**  \( \sum_{x_i \in Y} p_i \);

**Goal**  maximize
2. Complexity classes, reduction, and completeness

Complexity classes for decision problems

Let $\mathcal{A}$ be an algorithm running on a set of instances $I$. Let $x \in I$.

- $\hat{t}_\mathcal{A}(x) = \text{running time}$ of $\mathcal{A}$ on $x$ ($\approx$ number of elementary steps);
- $\hat{s}_\mathcal{A}(x) = \text{running space}$ of $\mathcal{A}$ on $x$ (size of memory needed to run $\mathcal{A}$ on $x$).
- the worst-case running time of $\mathcal{A}$ is the function $t_\mathcal{A} : \mathbb{N} \to \mathbb{N}$ defined by
  \[ t_\mathcal{A}(n) = \max\{\hat{t}_\mathcal{A}(x) \mid x \in I, |x| \leq n\} \]
- the worst-case running space of $\mathcal{A}$ is the function $s_\mathcal{A} : \mathbb{N} \to \mathbb{N}$ defined by
  \[ s_\mathcal{A}(n) = \max\{\hat{s}_\mathcal{A}(x) \mid x \in I, |x| \leq n\} \]
- the running time of $\mathcal{A}$ is in $O(g(n))$ if $t_\mathcal{A}(n)$ is $O(g(n))$.
- the running space of $\mathcal{A}$ is in $O(g(n))$ if $s_\mathcal{A}(n)$ is $O(g(n))$.

Refresher: given two functions $f, g : \mathbb{N} \to \mathbb{N}$, we say that $f(n)$ is $O(g(n))$ if there exist constants $c, a$ and $n_0$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n) + a$. 
2. Complexity classes, reduction, and completeness

**Complexity classes for decision problems**

\[ t_A(n) = \max \{ \hat{t}_A(x) \mid x \in I, |x| \leq n \} \]
\[ s_A(n) = \max \{ \hat{s}_A(x) \mid x \in I, |x| \leq n \} \]

|\(x| = \text{size of the input } |x|\)

**Important!** the running time and space of an algorithm is heavily dependent on how |\(x| is defined

|\(x| defined as the number of symbols (from a fixed alphabet) needed to encode \(x\).

**Examples**

- \(x = \text{graph } G = \langle V, E \rangle \Rightarrow |x| = |V| + |E|\).
- \(x = \text{propositional formula} \Rightarrow |x| = \text{length of } x\).
- \(x = \text{positive integer number} \Rightarrow |x| = (\text{usually}) \log_2 |x|\).
Example

Consider the following algorithm $\mathcal{A}$ for PRIME:

Input = $x$; \hspace{0.5cm} n = $|x| = \log_2 x$.

$y := 2$; $\text{prime} := \text{TRUE}$;

Repeat
  
  if $x$ is a multiple of $y$ then $\text{prime} := \text{FALSE}$;
  
  else $y := y + 1$

Until $y^2 > x$ or $\text{prime} = \text{FALSE}$

Checking whether $x$ is a multiple of $y$: runs in $O(\log_2 x)$.
Checking whether $y^2 > x$: runs in $O(\log_2 x)$ (because $y \leq x$).

$\mathcal{A}$ runs in $O(\sqrt{x} \log_2 x) \approx O(2^{n/2})$. 
Complexity classes for decision problems

A decision problem \( \mathcal{P} \) can be solved with time (resp. space) \( f(n) \) if there exist an algorithm \( \mathcal{A} \) that solves \( \mathcal{P} \) and whose running time (resp. space) is in \( O(g(n)) \).

For any function \( f \):

- \( \text{TIME}(f(n)) \) is the set of all decision problems which can be solved in time \( f(n) \).
- \( \text{SPACE}(f(n)) \) is the set of all decision problems which can be solved in space \( f(n) \).

Three first complexity classes:

**polynomial time** \( P = \bigcup_{k=0}^{\infty} \text{TIME}(n^k) \)

**polynomial space** \( \text{PSPACE} = \bigcup_{k=0}^{\infty} \text{SPACE}(n^k) \)

**exponential time** \( \text{EXPTIME} = \bigcup_{k=0}^{\infty} \text{TIME}(2^{n^k}) \)
Two properties:

- \( \text{TIME}(f(n)) \subseteq \text{SPACE}(f(n)) \)

- if \( f \leq g \) (from a given integer \( n_0 \)) then \( \text{TIME}(f(n)) \subseteq \text{TIME}(g(n)) \) and \( \text{SPACE}(f(n)) \subseteq \text{SPACE}(g(n)) \)

Consequence:

\[ \text{P} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \]

\( \text{P} \subseteq \text{PSPACE} \)? open (believed to be true)

\( \text{PSPACE} \subseteq \text{EXPTIME} \)? open (believed to be true)

However we know that \( \text{P} \subseteq \text{EXPTIME} \).
2. Complexity classes, reduction, and completeness

**Nondeterministic algorithm**: apart from all usual constructs, can execute commands of the type “*guess* $y \in \{0, 1\}$”.

Structure of a nondeterministic algorithm = *computation tree* (guess instructions corresponding to branching points)

$\neq$ linear structure of a deterministic algorithm (at any step, one possible next step)
2. Complexity classes, reduction, and completeness

Nondeterministic problem solution

\( \mathcal{P} = \langle I_\mathcal{P}, Y_\mathcal{P} \rangle \) decision problem.

A nondeterministic algorithm \( \mathcal{A} \) solves \( \mathcal{P} \) if, for all \( x \in I_\mathcal{P} \):

1. \( \mathcal{A} \) running on \( x \) halts for any possible guess sequence;

2. \( x \in Y_\mathcal{P} \) iff there exists a sequence of guesses which leads \( \mathcal{A} \) to return the value YES.

Example: nondeterministic algorithm for SAT:

Input: \( \varphi = C_1 \wedge \ldots \wedge C_n \)

guess an assignment \( f \) \( [= \) guess \( f(v_1), \ldots, f(v_n) \)]

if for each clause \( C_i, f \models C_i \)
then return YES
else return NO
2. Complexity classes, reduction, and completeness

Nondeterministic time and space

A nondeterministic algorithm $\mathcal{A}$ solves $P$ in time $t(n)$ iff

1. for all $x \in I_P$ with $|x| = n$, $\mathcal{A}$ halts for any guess sequence;

2. $x \in Y_P$ iff there exists a guess sequence which leads the algorithm to return YES in time at most $t(n)$.

$NTIME(f(n)) = \text{set of all decision problems solvable by a nondeterministic algorithm in time } O(f(n))$.

Similarly: $NSPACE(f(n)) = \text{set of all decision problems solvable by a nondeterministic algorithm in space } O(f(n))$.

**nondeterministic polynomial time** $\text{NP} = \bigcup_{k=0}^{\infty} NTIME(n^k)$

**Example**: $SAT \in NTIME(n) \subseteq \text{NP}$
2. Complexity classes, reduction, and completeness

**Relations between classes**

- $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$ and $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$
- $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$
- $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n + f(n)})$
- $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$ for any “proper” function $f$ such that $f(n) \leq \log n$ from a given $n_0$.

**Corollaries:**

(a) $\text{P} \subseteq \text{NP}$
(b) $\text{NP} \subseteq \text{PSPACE}$
(c) $\text{PSPACE} \subseteq \text{EXPTIME}$
(d) $\text{NPSPACE} = \text{PSPACE}$
2. Complexity classes, reduction, and completeness

**Reductions between problems**

$P_1, P_2$ two decision problems.

**Karp reducibility**: $P_1$ is *Karp-reducible* (or many-to-one reducible) to $P_2$ if there exists an algorithm $\mathcal{R}$ which given any instance $x \in I_{P_1}$ of $P_1$, transforms it into an instance $y \in I_{P_2}$ of $P_2$ in such a way that $x \in Y_{P_1}$ if and only if $y \in Y_{P_2}$. $\mathcal{R}$ is said to be a *Karp reduction* from $P_1$ to $P_2$ and we write $P_1 \leq_m P_2$. If both $P_1 \leq_m P_2$ and $P_2 \leq_m P_1$ then $P_1$ and $P_2$ are *Karp-equivalent*.

**polynomial-time Karp reducibility**: $P_1$ is polynomial-time Karp-reducible to $P_2$ if and only if $P_1$ is Karp-reducible to $P_2$ and the corresponding reduction $\mathcal{R}$ is a polynomial-time algorithm. We write $P_1 \leq_p^m P_2$.

Intuitions:

- $\mathcal{R}$ gives a method for solving $P_1$ using an algorithm for $P_2$;
- $P_2$ is at least as difficult as $P_1$;
- if $P_1 \leq_p^m P_2$ and $P_1$ can be solved efficiently then $P_1$ can be solved efficiently.
2. Complexity classes, reduction, and completeness

**Reductions between problems**

Karp reducibility applies only to decision problems.

**Oracles:** let $P = \langle I_P, S_P, R \rangle$ be a function problem. An oracle for $P$ is an abstract device which, for any $x \in I_P$, returns a value $f(x) \in S_P$ in just one computation step.

**Turing reducibility:** let $P_1, P_2$ two function problems. $P_1$ is Turing-reducible to $P_2$ if there exists an algorithm $R$ for $P_1$ which queries an oracle for $P_2$. $R$ is said to be a Turing reduction from $P_1$ to $P_2$. We write $P_1 \leq_T P_2$.

Karp-reducibility is a particular case of Turing-reducibility: case where

- $P_1$ and $P_2$ are decision problems (seen as particular cases of function problems);
- the oracle for $P_2$ is queried just once;
- $R$ returns the value answered by the oracle.

In general: Karp-reducibility is weaker than Turing-reducibility.

**polynomial-time Turing reducibility:** $P_1 \leq^p_T P_2$ iff $P_1 \leq_T P_2$ and the corresponding reduction $R$ is polynomial-time computable with respect to the size of the input.
Closure and completeness

A complexity class $C$ is closed with respect to a reducibility $\leq_r$ if for any pair of decision problems $P_1$ and $P_2$ such that $P_1 \leq_r P_2$, $P_2 \in C$ implies $P_1 \in C$.

$\implies$ P, NP, PSPACE and EXPTIME are all closed under $\leq^p_m$.

For any complexity class $C$, a decision problem $P$ is $C$-complete with respect to a reducibility $\leq_r$ if for any other decision problem $P' \in C$ we have $P' \leq_r P$. 
2. Complexity classes, reduction, and completeness

**NP-completeness**

In particular: a decision problem $\mathcal{P}$ is NP-complete (implicitly: for $\leq^p_m$) if $\mathcal{P} \in \text{NP}$ and for any decision problem $\mathcal{P}' \in \text{NP}$, we have $\mathcal{P}' \leq^p_m \mathcal{P}$.

**Examples:** SAT is NP-complete (Cook, 1971).

NP closed with respect to $\leq^p_m$

and $P \subseteq \text{NP}$

$\Rightarrow P = \text{NP}$ if and only if at least one NP-complete-problem is in $P$.

$P \subset \text{NP}$? open (believed to be true)

Thousands of known NP-complete problems

Consider a decision problem \( P = \langle I_P, Y_P \rangle \) involving positive integer numbers. Let \( \max(x) \) denote the highest number appearing in instance \( x \in I_P \). (Note that \( \max(x) \) can be exponential in \( |x| \).)

An algorithm \( \mathcal{A} \) for \( P \) is \textit{pseudopolynomial} if it is polynomial in \( |x| \) and in \( \max(x) \).

\( P \) is \textit{strongly NP-complete} if it is still NP-complete when the integers numbers appearing in the instances are encoded in \textit{unary notation}.

\textbf{Examples}

- \textsc{MAXSAT}(K) is strongly NP-complete
- \textsc{KNAPSACK}(K) is not strongly NP-complete (unless \( P = NP \)).

If \( P \) is strongly NP-complete, then there exists no pseudopolynomial algorithm that solves it, unless \( P = NP \).
A pseudopolynomial algorithm for \textsc{knapsack}(K).

\textsc{knapsack}(K):

\textbf{Instance} a set \(X = \{x_1, \ldots, x_n\}\) of objects; for each object, a volume \(a_i\) and a value \(p_i\); a maximum volume \(b\); and \(K\).

\textbf{Question} Is there a subset \(Y \subseteq X\) such that \(\sum_{i \in Y} a_i \leq b\) and \(\sum_{i \in Y} p_i \geq K\)?

\(V(w, i)\) = largest value attainable by selecting some among the first \(i\) items so that the total volume is exactly \(w\). \(V(w, i)\) can be computed by \textit{dynamic programming}:

\begin{verbatim}
for \(w := 1\) to \(b\) do \(V(w, 0) := 0\);
for \(i := 1\) to \(n\) do
    for \(w := 1\) to \(b\) do
        if \(a_i > w\) then \(V(w, i) := V(w, i - 1)\)
        else \(V(w, i) := \max[V(w, i - 1), p_i + V(w - a_i, i - 1)]\);
    if \(V(w, n) \geq K\) for some \(w\) then return \text{YES} else return \text{NO}.
\end{verbatim}

The algorithm runs in \(O(nb)\).
2. Complexity classes, reduction, and completeness

Complements of decision problems and classes

\( \mathcal{P} = \langle I_{\mathcal{P}}, Y_{\mathcal{P}} \rangle \) decision problem

\( \overline{\mathcal{P}} = \langle I_{\mathcal{P}}, I_{\mathcal{P}} \setminus Y_{\mathcal{P}} \rangle \) complement of \( \mathcal{P} \).

Example: \( \overline{\text{SAT}} \) set of all unsatisfiable propositional formulas.

If \( C \) is a complexity class then \( \text{co}C = \{ \overline{\mathcal{P}} \mid \mathcal{P} \in C \} \)

- if \( C = \text{TIME}(f(n)) \) then \( \text{co}C = C \)
- if \( C = \text{SPACE}(f(n)) \) then \( \text{co}C = C \)

\( \Rightarrow \) coP = P, coPSPACE = PSPACE

\( \Rightarrow \) coC is relevant only if \( C \) is a nondeterministic complexity class
2. Complexity classes, reduction, and completeness

The class coNP

\[ \text{coNP} = \{ \overline{P} \mid P \in \text{NP} \} \]

- \text{NP} class of problems that have succinct certificates
- \text{coNP} class of problems that have succinct disqualifications

if \( P \) is \( \text{NP} \)-complete then \( \overline{P} \) is \( \text{coNP} \)-complete.

Example: \( \text{SAT} \) is \( \text{NP} \)-complete \( \Rightarrow \) \( \text{UNSAT} \) and \( \text{VALIDITY} \) are \( \text{coNP} \).

- \( P \subseteq \text{coNP} \)
- \( P \subset \text{coNP} \) (believed to be true)
- \( \text{NP} = \text{coNP} \) (believed to be false)
- if \( P = \text{NP} \) then \( \text{NP} = \text{coNP} \)
- if \( \text{NP} \subseteq \text{coNP} \) then \( \text{NP} = \text{coNP} \)
3. Above $NP$ and $coNP$

**The class $BH_2$ (or $DP$)**

**EXACT TSP:**

**Instance** a complete valued graph $G = \langle V, D \rangle$ and an integer $K$.

**Question** is the length of the shortest tour of $G$ equal to $K$?

$\langle G, K \rangle$ is in **EXACT TSP**

if and only if  

(1) $\langle G, K \rangle$ is in TSP and (2) $\langle G, K - 1 \rangle$ is not in TSP

if and only if  

(1) $\langle G, K \rangle$ is in TSP and (2) $\langle G, K - 1 \rangle$ is in $\overline{\text{TSP}}$.

(1) in $NP$  (2) in $coNP$

EXACT TSP is the *intersection of a problem in $NP$ and a problem in $coNP$*

A decision problem $\mathcal{P} = \langle I, Y \rangle$ is in $BH_2$ (or $DP$) if and only if there exist two problems $\mathcal{P}_1 = \langle I, Y_1 \rangle \in NP$ and $\mathcal{P}_2 = \langle I, Y_2 \rangle \in coNP$ such that $Y = Y_1 \cap Y_2$. 
3. Above NP and coNP

**BH\(_2\) and coBH\(_2\)**

**SAT-UNSAT:**

**Instance** two propositional formulas \(\varphi\) and \(\psi\)

**Question** is \(\varphi\) satisfiable and \(\psi\) unsatisfiable?

SAT-UNSAT is BH\(_2\)-complete.

EXACT TSP is BH\(_2\)-complete.

\(\mathcal{P} = \langle I, Y \rangle \in \text{coBH}\(_2\) \iff\) there exist two problems \(\mathcal{P}_1 = \langle I, Y_1 \rangle \in \text{NP}\) and \(\mathcal{P}_2 = \langle I, Y_2 \rangle \in \text{coNP}\) such that \(Y = Y_1 \cup Y_2\).

**SAT-OR-UNSAT:**

**Instance** two propositional formulas \(\varphi\) and \(\psi\)

**Question** is \(\varphi\) satisfiable or \(\psi\) unsatisfiable?

SAT-OR-UNSAT is coBH\(_2\)-complete.
3. Above NP and coNP

The Boolean hierarchy

- BH$_0 = P$, and for every $k \geq 0$,
- BH$_{2k+1}$ is the class of all decision problems $P = \langle I, Y_1 \cap Y_2 \rangle$ such that $Y = Y_1 \cup Y_2$, where $\langle I, Y_1 \rangle \in BH_{2k}$ and $\langle I, Y_2 \rangle \in NP$;
- BH$_{2k+2}$ is the class of all decision problems $P = \langle I, Y_1 \cap Y_2 \rangle$ such that $Y = Y_1 \cap Y_2$, where $\langle I, Y_1 \rangle \in BH_{2k+1}$ and $\langle I, Y_2 \rangle \in NP$.

In particular:

$P = \langle I, Y \rangle$ is in BH$_3$ iff $Y = Y_1 \cap (Y_2 \cup Y_3)$, where $\langle I, Y_1 \rangle \in NP$, $\langle I, Y_2 \rangle \in NP$ and $\langle I, Y_3 \rangle \in coNP$.

NP, coNP $\subseteq$ BH$_2$, coBH$_2$ $\subseteq$ BH$_3$, coBH$_3$ ...
3. Above NP and coNP

Example: decision making in propositional logic with dichotomous preferences. Decision making context = 2 propositional formulas \( \varphi \) (integrity constraint), \( \gamma \) (goal).\n\( \vec{x} \) feasible decision if \( \vec{x} \models \varphi \), good decision if \( \vec{x} \models \varphi \land \gamma \), bad decision if \( \vec{x} \models \varphi \land \neg \gamma \), \( \vec{x}, \vec{y} \) two feasible decisions: \( \vec{x} \succ_{\varphi, \gamma} \vec{y} \) (\( \vec{x} \) better than \( \vec{y} \)) if \( \vec{x} \) is good and \( \vec{y} \) is bad.

COMPARISON:

Instance \( \varphi, \gamma, \) and \( \vec{x}, \vec{y} \) two truth assignments

Question Is \( \vec{x} \) better than \( \vec{y} \)?

NON-DOMINANCE:

Instance \( \varphi, \gamma, \) and \( \vec{x} \) truth assignment

Question Is it true that there exist no \( \vec{y} \) such that \( \vec{y} \) is better than \( \vec{x} \)?

CAND-OPT-SAT:

Instance \( \varphi, \gamma, \) and \( \psi \) propositional formula (expressing a given property)

Question Does there exist a non-dominated solution satisfying \( \psi \)?

Complexity of these three problems?
3. Above NP and coNP

**Oracles and relativized complexity classes**

\( A \) decision problem

\( C \) complexity class

\( C^A \) is the class of all decision problems that can be recognized with complexity \( C \) by an algorithm using oracles for \( A \).

*Example:* \( P^{\text{SAT}} \) is the class of all decision problems that can be solved in polynomial time using oracles for \( \text{SAT} \).

If \( C, C' \) are two complexity classes, \( C^{C'} \) is the class of all decision problems that can be recognized with complexity \( C \) by an algorithm using oracles for a \( C' \)-complete problem.

N.B. an oracle for \( A \) is an oracle for \( \overline{A} \Rightarrow C^{C'} = C^{\overline{C'}} \)

*Example:* \( P^{\text{NP}} = P^{\text{SAT}} \)
3. Above NP and coNP

The polynomial hierarchy

- $\Delta^p_1 = P$
- $\Sigma^p_1 = NP$
- $\Pi^p_1 = coNP$

and for every $k \geq 2$:
- $\Delta^p_k = P^{\Sigma^p_{k-1}}$
- $\Sigma^p_k = NP^{\Sigma^p_{k-1}}$
- $\Pi^p_k = co\Sigma^p_k$

- *polynomial hierarchy:* $PH = \cup_{k \leq 0} \Sigma^p_k = \cup_{k \leq 0} \Pi^p_k$

$k_{th}$ level of the polynomial hierarchy: $\{\Sigma^p_k, \Pi^p_k, \Delta^p_{k+1}\}$.

If $\Sigma^p_k = \Pi^p_k$ for some $k \geq 1$ then the polynomial hierarchy collapses at the $k_{th}$ level.

*Many interesting AI problems are located in the polynomial hierarchy.*
3. **Above NP and coNP**

\[ \Delta_2^P = P^{NP} \]

\( \Delta_2^P \) is the class of all decision problems that can be solved in polynomial time using NP-oracles.

**Remark:** co\( \Delta_2^P = ?? \)

\( \Theta_2^P = \Delta_2^P[O(\log n)] \) is the class of all decision problems that can be solved in polynomial time using at most a logarithmic number of NP-oracles.

**MAXSAT-PROPERTY:**

**Instance** \( \varphi = C_1 \land \ldots \land C_n \), and \( \psi \) a propositional formula (expressing a given property)

**Question** Does there exist an optimal assignment satisfying \( \psi \)?
3. Above NP and coNP

An algorithm for MAXSAT-PROPERTY:

1. \( k_{\text{min}} := 0; k_{\text{max}} := n; \)
2. Repeat
3. \( k := \left\lceil \frac{k_{\text{min}} + k_{\text{max}}}{2} \right\rceil; \)
4. if there exists an assignment satisfying at least \( k \) clauses of \( \varphi \)
5. then \( k_{\text{min}} := k \)
6. else \( k_{\text{max}} := k - 1 \)
7. Until \( k_{\text{min}} = k_{\text{max}}; \)
8. \( k^* = k_{\text{min}}(= k_{\text{max}}); \)
9. guess assignment \( \vec{x}; \)
10. check that \( \vec{x} \) satisfies \( k \) clauses of \( C; \)
11. check that \( \vec{x} \models \psi. \)

Step 4: oracle for MAXSAT(\( k \)) \( \Rightarrow \) NP-oracle

The algorithm works in deterministic polynomial time with \( O(\log n) \) oracles
\( \Rightarrow \) MAXSAT-PROPERTY is in \( \Delta_2^p[O(\log n)] \) (actually: \( \Delta_2^p[O(\log n)] \)-complete)
3. Above NP and coNP

WEIGHTED-SAT-PROPERTY:

**Instance** \( \varphi = \{ \langle C_1, w_1 \rangle, \ldots, \langle C_n, w_n \rangle \} \), where each \( C_i \) is a clause and each \( x_i \) is an integer, and \( \psi \) a propositional formula (expressing a given property).

**Question** Does there exist an optimal assignment (maximizing the sum of the weights of the satisfied clauses) satisfying \( \psi \)?

Show that MAXSAT-PROPERTY is in \( \Delta_2^P \) (actually: it is \( \Delta_2^P \)-complete).
3. Above NP and coNP

Second level of the polynomial hierarchy

$$\Sigma_2^p = \text{NP}^\text{NP}; \Pi_2^p = \text{coNP}^\text{NP}$$

Canonical complete problems for $\Sigma_2^p$ and $\Pi_2^p$: quantified Boolean formulas of depth 2.

QBF$_2,\exists$:

**Instance** a propositional formula $\varphi$ built on a set of propositional variables $V$, and a partition $\{A, B\}$ of $V$.

**Question** Does there exist an truth assignment $\bar{a}$ of the variables in $A$ such that for every truth assignment $\bar{b}$ of the variables in $B$, we have $(\bar{a}, \bar{b}) \models \varphi$?

(Short notation: $\exists A \forall B \varphi$)

Similarly: QBF$_2,\forall$: $\forall A \exists B \varphi$

- QBF$_2,\exists$ is $\Sigma_2^p$-complete
- QBF$_2,\forall$ is $\Pi_2^p$-complete
3. Above NP and coNP

SKEPTICAL INFERENCE:

**Instance** a set of propositional formulas $\Delta = \{\varphi_1, \ldots, \varphi_n\}$ and a formula $\psi$

**Question** Is it true that for every maximal consistent subset $S$ of $\Delta$ we have $\bigwedge S \models \psi$?

**Example:**

$\Delta = \{a \land b, b \rightarrow c, \neg b, a \land \neg c\}$;

maximal consistent subsets of $\Delta$:

$\{\{a \land b, b \rightarrow c\}, \{a \land b, a \land \neg c\}, \{b \rightarrow c, \neg b, a \land \neg c\}\}$

$a$ is skeptically inferred from $\Delta$. 
3. Above NP and coNP

**SKEPTICAL INFERENCE** is in $\Sigma_2^p$:

1. a subset $S$ of $\Delta$.
2. **for every** $\varphi_i \in \Delta \setminus S$:
   3. check that $S \cup \{\varphi_i\}$ is unsatisfiable;
   4. check that $\bigwedge S \not\models \psi$

Works in nondeterministic polynomial time with NP oracles

$\Rightarrow$ **SKEPTICAL INFERENCE** is in $\Sigma_2^p$

$\Rightarrow$ **SKEPTICAL INFERENCE** is in $\Pi_2^p$

Show that **SKEPTICAL INFERENCE** is $\Pi_2^p$-complete
3. Above NP and coNP

\[ QBF_{k, \exists} : \begin{cases} 
\text{if } k \text{ odd} & \exists X_1 \forall X_2 \ldots \exists X_k \varphi \\
\text{if } k \text{ even} & \exists X_1 \forall X_2 \ldots \forall X_k \varphi 
\end{cases} \]

\[ QBF_{k, \forall} : \begin{cases} 
\text{if } k \text{ odd} & \forall X_1 \exists X_2 \ldots \forall X_k \varphi \\
\text{if } k \text{ even} & \forall X_1 \exists X_2 \ldots \exists X_k \varphi 
\end{cases} \]

- \( QBF_{k, \exists} \) is \( \Sigma^p_k \)-complete
- \( QBF_{k, \forall} \) is \( \Pi^p_k \)-complete
3. Above NP and coNP

Polynomial space

PSPACE = class of all decision problems that can be solved in deterministic time using a polynomial amount of space.

QBF = set of all $\text{QBF}_{k,\exists}$ and $\text{QBF}_{k,\forall}$ for $k \in \mathbb{N}$

QBF is PSPACE-complete

$\text{PH} \subseteq \text{PSPACE}$

$\text{PH} = \text{PSPACE}$? open (believed to be false)
4. Complexity classes for function and optimization problems

FP and FNP

\( \mathcal{P} = \langle I_\mathcal{P}, S_\mathcal{P}, R \rangle \) function (search) problem

Solving \( \mathcal{P} \): establish, given an \( x \), whether there exists a \( s \) such that \( R(x, s) \) holds, and if so, give one possible value for that \( s \).

- \( \mathcal{P} \) is in FP if and only if there is a deterministic polynomial time algorithm that, given \( x \), can find some \( s \) such that \( R(x, s) \) holds, or return NO otherwise.
- \( \mathcal{P} \) is in FNP if and only if there is a deterministic polynomial time algorithm that, given \( x \) and \( s \), can determine whether \( R(x, s) \) holds.

Decision problem \( \mathcal{P}_D \) induced by a function problem \( \mathcal{P} \): \( \mathcal{P}_D = \langle I_\mathcal{P}, Y_\mathcal{P} \rangle \) with

\[
Y_\mathcal{P} = \{ x \mid R(x, s) \text{ holds for some } s \}
\]

FP = FNP if and only if \( \mathcal{P} = \text{NP} \).
4. Complexity classes for function and optimization problems

\( \mathcal{P} \) is always at least as hard as \( \mathcal{P}_D \). Can \( \mathcal{P} \) be significantly harder than \( \mathcal{P}_D \)?

Sometimes no:

\( \text{FSAT} \) can be solved in polynomial time iff \( \text{SAT} \) can be solved in polynomial time

\( \Rightarrow \) obvious

\( \Leftarrow \) let \( \varphi \) built on variables \( \{x_1, \ldots, x_n\} \).

1. if \( \varphi \) is unsatisfiable
2. then return \( \text{NO} \)
3. else for \( i := 1 \) to \( n \) do
4. if \( \varphi_{x_i} := \text{TRUE} \) is satisfiable
5. then \( f(x_i) := \text{TRUE}; \varphi := \varphi \land x_i \)
6. else \( f(x_i) := \text{FALSE}; \varphi := \varphi \land \neg x_i \)
7. end

If \( \varphi \) is satisfiable then the assignment \( f \) returned by the algorithm satisfies \( \varphi \)
4. Complexity classes for function and optimization problems

\( \mathcal{P} \) is always at least as hard as \( \mathcal{P}_D \). Can \( \mathcal{P} \) be significantly harder than \( \mathcal{P}_D \)?

Sometimes yes:

**ANOTHER HAMILTONIAN CYCLE:**

**Instance** a graph \( G \) that possesses a Hamiltonian cycle, and a Hamiltonian cycle \( C \) for \( G \).

**Question** does \( G \) have another Hamiltonian cycle?

A *cubic graph* is a graph \( G = \langle V, E \rangle \) where each vertex has degree 3.

**Proposition** (Papadimitriou, 93): given a cubic graph \( G \), and two vertices \( i, j \) of \( G \), the number of Hamiltonian cycles using \( (i, j) \) is even.

Corollary: if we know a Hamiltonian cycle for a cubic graph, *there must be another one*.

- **ANOTHER HAMILTONIAN CYCLE IN CUBIC GRAPHS** can be solved in unit time!
- **F-ANOTHER HAMILTONIAN CYCLE IN CUBIC GRAPHS** is not known to be in \( \mathcal{P} \).
4. Complexity classes for function and optimization problems

A function problem $\mathcal{P} = \langle I_{\mathcal{P}}, S_{\mathcal{P}}, R \rangle$ is **total** iff for every $x \in I_{\mathcal{P}}$ there exists an $s \in S_{\mathcal{P}}$ such that $R(x,s)$ holds.

TFNP $\subseteq$ FNP is the class of all total function problems that are in FNP.

Example: F-ANOTHER HAMILTONIAN CYCLE IN CUBIC GRAPHS is in TFNP.
4. Complexity classes for function and optimization problems

NPO

An optimization problem \( P = \langle I, SOL, m, \text{goal} \rangle \) is in NPO if and only if

1. the set of instances \( I \) is recognizable in polynomial time;

2. there exists a polynomial \( q \) such that for any \( x \in I \):
   - for any \( s \in SOL(x), |s| \leq q(|x|) \);
   - for any \( s \) such that \( |s| \leq q(|x|) \), it is decidable in polynomial time whether \( s \in SOL(x) \);

3. \( m \) is computable in polynomial time.

Example: MAXSAT is in NPO
4. Complexity classes for function and optimization problems

\[ P = \langle I, SOL, m, \text{goal} \rangle \] optimization problem \( \mapsto \) three induced problems:

**Constructive problem** \( P_C \) given an instance \( x \in I \), derive an optimal solution \( s^*(x) \in SOL^*(x) \) and its measure \( m^*(x) \).

**Evaluation problem** \( P_E \) given an instance \( x \in I \), derive the value \( m^*(x) \).

**Decision problem** \( P_D \) given an instance \( x \in I \) and a positive integer \( K \), decide whether \( m^*(x) \geq K \) (if \( \text{goal} = \text{MAX} \)) or whether \( m^*(x) \leq K \) (if \( \text{goal} = \text{MIN} \)).

**Underlying language of** \( P \):
- \( \{ (x, K) \mid x \in I \text{ and } m^*(x) \geq K \} \) if \( \text{goal} = \text{MAX} \)
- \( \{ (x, K) \mid x \in I \text{ and } m^*(x) \leq K \} \) if \( \text{goal} = \text{MIN} \)

For any optimization problem \( P \) in \( \text{NPO} \), the corresponding decision problem \( P_D \) belongs to \( \text{NP} \).
An optimization problem belongs to PO if it is in NPO and there exists a polynomial-time algorithm $A$ that, for any instance $x \in I$, returns an optimal solution $s \in SOL^*(x)$, together with its value $m^*(x)$.

*Example:* MINIMUM PATH is in PO.
4. Complexity classes for function and optimization problems

An optimization problem \( \mathcal{P} \) is NP-hard if for every decision problem \( \mathcal{P}' \in \text{NP} \), we have \( \mathcal{P}' \leq^p_{T} \text{SOL}^*_\mathcal{P} \), that is, \( \mathcal{P}' \) can be solved in polynomial time by an algorithm using an oracle that, for any instance \( x \in I_\mathcal{P} \), returns an optimal solution \( s^*(x) \) of \( x \) together with its value \( m^*_\mathcal{P}(x) \).

Results:

1. For any problem \( \mathcal{P} \in \text{NPO} \): if the underlying language of \( \mathcal{P} \) is NP-complete, then \( \mathcal{P} \) is NP-hard.

2. If \( \mathcal{P} \neq \text{NP} \) then \( \text{PO} \neq \text{NPO} \).

Example: MAXSAT is NP-hard.
4. Complexity classes for function and optimization problems

\[ \mathcal{P} = \langle I, SOL, m, \text{goal} \rangle \] optimization problem

\[ \mathcal{P}_C \] induced constructive problem; \( \mathcal{P}_E \) induced evaluation problem; \( \mathcal{P}_D \) induced decision problem.

- For any \( \mathcal{P} \in \text{NPO} \) we have \( \mathcal{P}_D \equiv_T^p \mathcal{P}_E \leq_T^p \mathcal{P}_C \)
- Let \( \mathcal{P} \in \text{NPO} \). If \( \mathcal{P}_D \) is \text{NP}-complete, then \( \mathcal{P}_C \leq_T^p \mathcal{P}_D \)

\( \Rightarrow \) when the decision problem \( \mathcal{P}_D \) is \text{NP}-complete then the constructive, evaluation, and decision problems are equivalent.

Question: is there an \text{NPO} problem \( \mathcal{P} \) whose corresponding constructive problem is harder than the evaluation problem \( \mathcal{P}_E \)? \text{open} (believed to be true).
5. Approximation

Approximation

When facing a NP-hard optimization problem: try to find a polynomial approximation algorithm.

Given an optimization problem \( \mathcal{P} = \langle I, SOL, m, \text{goal} \rangle \), an algorithm \( \mathcal{A} \) is an approximation algorithm for \( \mathcal{P} \) if, for any instance \( x \in I \), it returns an approximate solution, that is, a feasible solution \( \mathcal{A}(x) \in SOL(x) \).

Two criteria for evaluating the quality of an approximation algorithm \( \mathcal{A} \):

- the complexity of \( \mathcal{A} \).
- the quality of the approximate solution returned by \( \mathcal{A} \):
  - absolute approximation;
  - relative approximation;
  - differential approximation.
5. Approximation

**Absolute approximation**  $\mathcal{P}$ optimization problem; $x \in I$; $s \in SOL(x)$.

The *absolute error* of $s$ with respect to $x$ is defined as

$$D(x, s) = |m^*(x) - m(x, s)|$$

An approximation algorithm $\mathcal{A}$ for $\mathcal{P}$ is an *absolute approximation algorithm* if there exists a constant $k$ such that for every instance $x$ of $\mathcal{P}$, $D(x, \mathcal{A}(x)) \leq k$.

Polynomial-time absolute approximation algorithms for NP-hard optimization problems are rare.
Relative approximation  \( P \) optimization problem; \( x \in I; s \in SOL(x) \).

The *relative error* of \( s \) with respect to \( x \) is defined as
\[
E(x, s) = \frac{|m^*(x) - m(x, s)|}{\max(m^*(x), m(x, s))}
\]

An approximation algorithm \( \mathcal{A} \) for \( P \) is an \( \epsilon \)-approximate algorithm if for every instance \( x \) of \( P \), \( E(x, \mathcal{A}(x)) \leq \epsilon \).

The *performance ratio* of \( s \) with respect to \( x \) is defined as
\[
R(x, s) = \max \left( \frac{m(x, s)}{m^*(x)}, \frac{m^*(x)}{m(x, s)} \right)
\]

\( E(x, s) = 1 - 1/R(x, s) \).

An approximation algorithm \( \mathcal{A} \) for \( P \) is an \( r \)-approximate algorithm if for every instance \( x \) of \( P \), \( R(x, \mathcal{A}(x)) \leq r \).

A NP-hard optimization problem \( P \) is \( \epsilon \)-approximable (resp. \( r \)-approximable) if there exists a polynomial-time \( \epsilon \)-approximate (resp. \( r \)-approximate) algorithm for \( P \).
5. Approximation

Differential approximation  \( P \) optimization problem; \( x \in I; s \in SOL(x) \).

Let \( m_*(x) \) be the measure of the worst solution for \( x \):

\[
 m_*(x) = \begin{cases} 
 \min \{m(s) | s \in SOL(x)\} & \text{if goal } = \max \\
 \max \{m(s) | s \in SOL(x)\} & \text{if goal } = \min 
\end{cases}
\]

Differential performance ration of \( s \) wrt \( x \):

\[
 \Delta(x, s) = \frac{|m_*(x) - m(x, s)|}{|m_*(x) - m_*(x)|}
\]

An differential performance ratio of an approximation algorithm \( \mathcal{A} \) for \( P \) is

\[
 \Delta_P(\mathcal{A}) = \min \{\Delta(x, \mathcal{A}(x)) | x \in I_P\}
\]

Differential approximation is unsensitive to the addition of constants to the measure function.
5. Approximation

2-approximation for MAXSAT

Input: \( C = \{C_1, \ldots, C_n\} \) on a set of variables \( V = \{v_1, \ldots, v_p\} \)

for all \( v_i \in V \) do \( f(v_i) = \text{TRUE} \)

repeat
  \( l := \) literal appearing in the maximum number of clauses in \( C \);
  remove from \( C \) the clauses containing \( l \);
  delete \( \neg l \) from all clauses containing \( \neg l \);
  remove all empty clauses from \( C \);
  if \( l = \neg v_i \) then \( f(v_i) = \text{FALSE} \);
until \( C = \emptyset \);

Return \( f \).

Show that this algorithm is a polynomial-time 2-approximate algorithm for MAXSAT.

\( \Rightarrow \) MAXSAT is 2-approximable
5. Approximation

2-approximation for MAXIMUM KNAPSACK

Idea: rank objects such that (without loss of generality) \( \frac{p_1}{a_1} \geq \frac{p_2}{a_2} \geq \ldots \geq \frac{p_n}{a_n} \)

Algorithm \( \mathcal{A}_1 \):

\[
Y := \emptyset;
\]

\[
\text{for } i := 1 \text{ to } n \text{ do}
\]

\[
\text{if } b \geq a_i \text{ then } b := b - a_i
\]

\[
\text{return } Y
\]

Is \( \mathcal{A}_1 \) an \( r \)-approximate for MAXIMUM KNAPSACK?

Algorithm \( \mathcal{A}_2 \):

\[
\text{run } \mathcal{A}_1;
\]

\[
p_{\text{max}} = \max_{i=1,\ldots,n} p_i = p_{i^*};
\]

\[
\text{if } p_{\text{max}} > \sum_{x_i \in Y} p_i \text{ then return } \{x_{i^*}\} \text{ else return } Y
\]

\( \mathcal{A}_2 \) is a 2-approximate algorithm for MAXIMUM KNAPSACK
5. Approximation

APX

APX is the class of all NPO problems \( \mathcal{P} \) such that there exists a polynomial-time \( r \)-approximate algorithm for \( \mathcal{P} \), for some \( r \geq 1 \).

Example

MAXSAT and MAXIMUM KNAPSACK are 2-approximable

\[ \Rightarrow \text{MAXSAT and MAXIMUM KNAPSACK are in APX.} \]
5. Approximation

If MIN-TSP is an APX then $P = NP$

Proof: consider the following reductions from HAMILTONIAN CIRCUIT to MIN TSP

$G = \langle V, E \rangle$ instance of HAMILTONIAN CIRCUIT, with $|V| = n$

For any $r \geq 1$: $F_r(G)$ instance of MIN TSP defined by the same graph $G$ and the distance function

$$d(v, v') = \begin{cases} 
1 & \text{if } (v, v') \in E \\
1 + n.r & \text{otherwise}
\end{cases}$$

- if $G$ has an Hamiltonian circuit then $m^*(F_r(G)) = n$;
- if $G$ has no Hamiltonian circuit then $m^*(F_r(G)) \geq n - 1 + (1 + n.r) = n(1 + r)$

Suppose there exists an $r$-approximate algorithm $\mathcal{A}$ for MIN TSP. Running it on $F(G)$ and returning YES if the instance returned by $\mathcal{A}$ has measure $n$ and NO otherwise provides a polynomial-time algorithm for HAMILTONIAN CIRCUIT

$\Rightarrow P = NP$
5. Approximation

Generalization: the “gap technique”

Given some NPO minimization problem \( P \), if there exists a constant \( k \) such that it is NP-hard to decide whether \( m^*(x) \leq k \), then \( P \) is not approximable for \( r < \frac{k+1}{k} \), unless \( P = \text{NP} \).
5. Approximation

Polynomial-time approximation schemes

Let \( \mathcal{P} \) be an optimization problem in \( \text{NPO} \). \( \mathcal{A} \) is a polynomial-time approximation scheme (PTAS) for \( \mathcal{P} \) if for any instance \( x \) of \( \mathcal{P} \) and any \( r > 1 \), \( \mathcal{A} \) applied to input \((x, r)\) returns an \( r \)-approximate solution of \( x \) in time polynomial in \(|x|\).

Example: PTAS for MIN PARTITION

MINIMUM PARTITION:

Instance a collection of objects \( X = \{x_1, \ldots, x_n\} \) with associated integer weights \( a_1, \ldots, a_n \);

Solution a partition of \( X \) into two sets \( Y \) and \( Z \);

Measure \( \max(\sum_{x_i \in Y} a_i, \sum_{x_i \in Z} a_i) \)

Goal minimize
5. Approximation

Let $X$ be an instance of MINIMUM PARTITION and $r$ such that (without loss of
generality) $1 < r < 2$.

Algorithm $\mathcal{A}_r$:

$k(r) := \lceil \frac{2-r}{r-1} \rceil$; reorder the $x_i$'s such that (wlog) $a_1 \geq \ldots \geq a_n$;
find an optimal partition $\{Y_1, Y_2\}$ of $\{x_i, 1 \leq i \leq k(r)\}$;
for $j := k(r) + 1$ to $n$ do
    if $\sum_{x_i \in Y_1} a_i \leq \sum_{x_i \in Y_2} a_i$
        then $Y_1 := Y_1 \cup \{x_j\}$
        else $Y_2 := Y_2 \cup \{x_j\}$

For any $r$, $\mathcal{A}_r$ is a $r$-approximate algorithm for MIN PARTITION.
5. Approximation

PTAS is the class of NPO problems that admit a polynomial-time approximation scheme

*Example*: MINIMUM PARTITION is in PTAS

- PTAS ⊆ APX;
- if P ≠ NP, then PTAS ⊂ APX.

Problem: while being polynomial in |x|, the running time of a polynomial-time approximation scheme may depend on $\frac{1}{r-1} \Rightarrow$ the better the approximation, the larger may be the running time.

*Example* the above scheme $\mathcal{A}_r$ for MINIMUM PARTITION runs in $O(2^{\frac{1}{r-1}})$. 
5. Approximation

**Fully polynomial-time approximation schemes**

Let $\mathcal{P}$ be an optimization problem in NPO. $\mathcal{A}$ is a **fully polynomial-time approximation scheme** (FPTAS) for $\mathcal{P}$ if for any instance $x$ of $\mathcal{P}$ and any $r > 1$, $\mathcal{A}$ applied to input $(x, r)$ returns an $r$-approximate solution of $x$ in time polynomial both in $|x|$ and in $\frac{1}{r-1}$.

FPTAS is the class of NPO problems that admit a polynomial-time approximation scheme.

*Example*: MAXIMUM KNAPSACK is in PTAS

An optimization problem is *polynomially bounded* if there exists a polynomial $p$ such that for any instance $x$ and any solution $s \in SOL(x)$, $m(x, s) \leq p(|x|)$.

No NP-hard polynomially bounded optimization problem is in FPTAS, unless $P = NP$.

- $\text{FPTAS} \subseteq \text{PTAS}$;
- if $P \neq \text{NP}$, then $\text{FPTAS} \subset \text{PTAS}$.
Compact representation languages: expressivity, compilability, succinctness

Compact representation languages / frames: tools for expressing succinctly some data concerning a combinatorial set of objects.

A compact representation frame is a 4-uple

$$\mathcal{L} = \langle D, \mathcal{G}, L, F \rangle$$

where

- $D = D_1 \times \ldots \times D_n$ object domain associated with a set of variables $X_1, \ldots, X_n$ with finite domains $D_1, \ldots, D_n$.
- $\mathcal{G}$ set of mathematical objects on $D$.
  For instance: set of all subsets of $D$; set of all functions from $D$ to $\mathcal{R}$; set of all probability distributions on $D$; set of all preorders on $D$; etc.
- $L$ compact representation language;
- $F : L \rightarrow \mathcal{G}$. 
5. Compact representation and compilation

What can we expect from a compact representation frame?

- $\mathcal{L}$ should be expressive: ideally, any object of $\mathcal{G}$ should be expressible in $\mathcal{L}$ (i.e. $F$ should be surjective);

- $\mathcal{L}$ should be as compact as possible. $\text{ExplRep}(\alpha) =$ “explicit” representation of $\alpha$: always in $O(2^n)$. We want not only $|F(\alpha)| \leq |\text{ExplRep}(\alpha)|$ but ideally, $|F(\alpha)| \ll |\text{ExplRep}(\alpha)|$ should hold for “most” instances $\alpha$ as possible;

- $F$ should be easily computable (ideally, $F(\alpha)$ computable in time polynomial in $|\alpha|$);

- relevant reasoning and decision making tasks should be as easy as possible when the input is expressed by the compact form $F(\alpha)$. 
5. Compact representation and compilation

Examples of compact representation languages

CONSTRAINTS:

- \( D = D_1 \times \ldots \times D_n \) (\( D_i \) finite domain of variables \( X_i \));
- \( G = 2^D \);
- \( L = \) set of CSPs over \( \{X_1, \ldots, X_n\} \);
- for any \( C \in L \), \( F(C) = \) set of solutions of \( C \).

PROPOSITIONAL LOGIC:

- \( D = \{0, 1\}^N \) set of propositional interpretations for propositional variables \( \{p_1, \ldots, p_n\} \);
- \( G = 2^D \);
- \( L = \) set of propositional formulas built on \( \{p_1, \ldots, p_n\} \);
- for any \( \varphi \in L \), \( F(\varphi) = \) set of models of \( \varphi \).
5. Compact representation and compilation

WEIGHTED FORMULAS:

• \( D = \{0, 1\}^N \) set of *propositional interpretations* for propositional variables \( \{p_1, \ldots, p_n\} \);

• \( G = (D \rightarrow \mathbb{Z}) \);

• \( L \): set of weighted formulas

\[
\Gamma = \{ \langle \varphi_1, w_1 \rangle, \ldots, \langle \varphi_p, w_p \rangle \}
\]

where the \( w_i \) are integers.

• for any \( \Gamma \in L \) and any \( s \in D \), \( F(\Gamma) \) is the utility function defined by

\[
F(\Gamma)(s) = u_\Gamma(s) = \sum \{ w_i \mid 1 \leq i \leq p, s \models \varphi_i \}
\]
5. Compact representation and compilation

**Bayesian networks:**

- $D = D_1 \times \ldots \times D_n$ ($D_i$ finite domain of variables $X_i$);
- $G = \text{set of probability distributions on } D$;
- $L = \text{set of Bayesian networks on } D$. A Bayesian network on $D$ is a pair $\mathcal{N} = \langle G, \{CPT(X_i), i = 1, \ldots, n\} \rangle$

where $G$ is a directed acyclic graph on $\{X_1, \ldots, X_n\}$ and for each $X_i$, $CPT(X_i)$ is a conditional probability table that specifies $p(d_i|d_{j_1}, \ldots, d_{j_p})$ for each $(d_{j_1}, \ldots, d_{j_p}) \in D_{j_1} \times \ldots D_{j_p}$, where $\text{Par}_G(X_i) = \{X_{j_1}, \ldots, X_{j_p}\}$. (If $X_i$ has no parents in $G$ then $CPT(X_i)$ is an unconditional probability distribution on $D_i$.)

- for any $\mathcal{N} \in L$ and any $\vec{d} \in D$, $F(\mathcal{N})$ is the probability distribution on $D$ defined by

$$F(\mathcal{N})(\vec{d}) = \prod_{i=1}^{n} p(d_i|(d_j| X_j \in \text{Par}_G(X_i))$$
5. Compact representation and compilation

CP-NETS (Boutilier, Brafman, Hoos and Poole, 99):

- \( D = D_1 \times \ldots \times D_n \) (\( D_i \) finite domain of variables \( X_i \));
- \( \mathcal{G} \) set of partial preorders (reflexive and transitive relations) on \( D \);
- \( L = \) set of CP-nets over \( D \). A CP-net over \( D \) is a pair \( \mathcal{N} = \langle \mathcal{G}, \{CPT(X_i), i = 1, \ldots, n\} \rangle \), where \( \mathcal{G} \) is a directed acyclic graph on \( \{X_1, \ldots, X_n\} \) and for each \( X_i \), \( CPT(X_i) \) is a conditional preference table that specifies a linear order on \( D_i \) for each \( (d_{j_1}, \ldots, d_{j_p}) \in D_{j_1} \times \ldots \times D_{j_p} \), where \( \text{Par}_G(X_i) = \{X_{j_1}, \ldots, X_{j_p}\} \). (If \( X_i \) has no parents in \( \mathcal{G} \) then \( CPT(X_i) \) is an unconditional preference relation on \( D_i \).)
- for any \( \mathcal{N} \in L \) and any \( \vec{d} \in D \), \( \succ_{\mathcal{N}} \) is the preference relation on \( D \) induced from \( \mathcal{N} \).
**CP-nets: example**

\[ x \succ \bar{x} \]

\[ x : y \succ \bar{y} \]

\[ \bar{x} : \bar{y} \succ y \]

\[ x \lor y : z \succ \bar{z} \]

\[ \neg(x \lor y) : \bar{z} \succ z \]

**if** \( X = x \)

then \( Y = y \) preferred to \( Y = \bar{y} \)

everything else (\( z \)) being equal (ceteris paribus)

\( xyz \succ x\bar{y}z; \quad xy\bar{z} \succ x\bar{y}\bar{z}; \)

\( \bar{x}\bar{y}z \succ \bar{x}yz; \quad \bar{x}\bar{y}\bar{z} \succ \bar{x}y\bar{z} \)
$CP-nets$: example

\[
\begin{align*}
x & \succ x \\
x \cdot y & \succ \bar{y} \\
\bar{x} & \succ y \\
x \lor y : z & \succ \bar{z} \\
\neg (x \lor y) & : \bar{z} \succ z
\end{align*}
\]

$\succ^X$: $xyz \succ \bar{xyz}$, $xy\bar{z} \succ \bar{xy}\bar{z}$, $x\bar{y}z \succ \bar{x}\bar{y}z$, $x\bar{y}\bar{z} \succ \bar{x}\bar{y}\bar{z}$

$\succ^Y$: $xyz \succ x\bar{y}z$, $xy\bar{z} \succ \bar{x}\bar{y}z$, $x\bar{y}z \succ \bar{x}yz$, $x\bar{y}\bar{z} \succ \bar{x}y\bar{z}$

$\succ^Z$: $xyz \succ xy\bar{z}$, $x\bar{y}z \succ \bar{x}y\bar{z}$, $x\bar{y}z \succ \bar{x}yz$, $x\bar{y}\bar{z} \succ \bar{x}y\bar{z}$

$\succ_C = \text{transitive closure of } \succ^X \cup \succ^Y \cup \succ^Z$
**CP-nets: example**

\[
\begin{align*}
&x \succ \bar{x} \\
&x : y \succ \bar{y} \\
&\bar{x} : \bar{y} \succ y \\
&x \lor y : z \succ \bar{z} \\
&\neg(x \lor y) : \bar{z} \succ z
\end{align*}
\]
5. Compact representation and compilation

Properties of compact representation languages: expressivity

Given $L = \langle D, G, L, F \rangle$, what is the subset of $G$ expressible in $L$?

$$Expr(L) = F(L) = \{ F(\alpha) \mid \alpha \in L \} \subseteq G$$

- $L$ is fully expressive iff $Expr(L) = G$;
- given $L_1, L_2$ with same $G$, $L_1$ is at least as expressive as $L_2$ iff $Expr(L_1) \supseteq Expr(L_2)$.

Examples:

- PROPOSITIONAL LOGIC, CNF, DNF are fully expressive;
- 2-CNF, HORN-CNF, 2-DNF are not fully expressive; none of them is more expressive than the other ones (all three are pairwise incomparable).
5. Compact representation and compilation

Expressivity (continued)

- **WEIGHTED FORMULAS**: fully expressive for the set of functions from $D$ to $\mathbb{Z}$;
- **WEIGHTED $k$-CLAUSES** and **WEIGHTED $k$-CUBES** are not fully expressive; they are as expressive as each other (provided that negative weights are allowed).

A utility function $u : 2^{\{p_1, \ldots, p_n\}} \to \mathbb{Z}$ is $k$-additive if

$$u(s) = \sum_{X \subseteq \{p_1, \ldots, p_k\}, |X| = k} u_X(s \cap X)$$

where for every $X$, $u_X : 2^X \to \mathbb{Z}$.

Remarks: (a) any $u$ is $n$-additive; (b) $u$ is 1-additive iff $u$ is linear: $u(s) = \sum_{p_i \in s} v_i$.

A simple result (Chevaleyre, Endriss & Lang 06):

1. $u$ is expressible in the language of **WEIGHTED $k$-CLAUSES** iff $u$ is $k$-additive;
2. $u$ is expressible in the language of **WEIGHTED $k$-CUBES** iff $u$ is $k$-additive.
5. Compact representation and compilation

Properties of compact representation languages: **succinctness** (Gogic, Kautz, Papadimitriou and Selman, 95; Cadoli, Donini, Liberatore and Schaerf, 96).

\[ L_1 = \langle D, G, L_1, F_1 \rangle; \quad L_2 = \langle D, G, L_2, F_2 \rangle, \]

\( L_1 \) is at least as succinct as \( L_2 \) (denoted \( L_1 \leq_S L_2 \)) iff there exists a function \( H \) from \( L_2 \) to \( L_1 \) such that:

1. \( \alpha \) and \( H(\alpha) \) are equivalent: for any \( \alpha \in L_2 \), \( F_1(H(\alpha)) = F_2(\alpha) \);

2. \( H \) is polysize: there exists a polynomial \( p \) such that for every \( \alpha \in L_2 \),

\[ |H(\alpha)| \leq p(|\alpha|). \]

Remarks:

- \( \leq_S \) is transitive;

- \( L_1 \leq_S L_2 \Rightarrow L_1 \) at least as expressive as \( L_2 \)
5. Compact representation and compilation

Compilation (preprocessing)

How can we deal with untractability?

- focus on tractable fragments (⇒ loss of expressivity)
- use approximate (e.g. randomized) algorithms (⇒ loss of optimality)
- separate the problem between a fixed part (known off-line) and a varying part (known only on-line) and preprocess the fixed part ⇒ compilation

Key motivation behind compilation: push as much of the computational burden into the on-line phase, which is then amortized over all on-line queries.
5. Compact representation and compilation

Examples:

DIAGNOSIS:

fixed part  the description of the system (how the components work, how inputs are related to outputs)

varying part  the observations

question  what are the faulty components?

KNOWLEDGE BASE QUERYING:

fixed part  the knowledge base Σ

varying part  the query Q (e.g., a clause)

question  is Q entailed by Σ?
5. Compact representation and compilation

Knowledge compilation (Darwiche and Marquis, 02)

The rest of this talk is based on the following paper:


Propositional reasoning is intractable $\Rightarrow$ preprocessing (when possible) is a good idea.

Problem: find an efficient target language into which the propositional knowledge base $\Sigma$ will be compiled.

Criteria for evaluating a target language:

1. its succinctness;
2. the class of queries that can be answered in polynomial time;
3. the class of transformations that can be applied to the representation in polynomial time.
Let \( \varphi \) be a propositional formula.

**negation normal form (NNF)** \( \varphi \) is in NNF iff every occurrence of the negation symbol in \( \varphi \) has only a propositional variable in its scope.

Tree representation of a formula NNF

\[
\varphi = ((A \leftrightarrow \neg B) \land (C \leftrightarrow D)) \lor ((A \leftrightarrow B) \land (C \leftrightarrow \neg D))
\]
**decomposability**  An NNF $\varphi$ is *decomposable* if for each conjunction $C$ in $\varphi$, the conjuncts of $\varphi$ do not share variables.

DNNF: subset of NNF satisfying decomposability.
determinism  An NNF $\phi$ is deterministic if for each disjunction $D$ in $\phi$, each two disjuncts of $D$ are logically contradictory.

\[ \neg A \quad B \quad \neg B \quad A \quad C \quad \neg D \quad D \quad \neg C \]

d-DNF: subset of NNF satisfying determinism;
d-DDNF: subset of NNF satisfying decomposability and determinism
5. Compact representation and compilation

**smoothness** An NNF $\varphi$ is *smooth* if for each disjunction $D$ in $\varphi$, each disjunct of $D$ mentions the same variables.

$s$–NNF: subset of NNF satisfying smoothness;

sd–DNNF: subset of NNF satisfying decomposability, determinism and smoothness.
Let $\varphi$ be a formula in $\text{NNF}$. $\varphi$ satisfies

**flatness** if its depth is at most 2.

**simple conjunction** if the children of or-nodes are leaves that share no variables (the node is a *clause*).

**simple disjunction** if the children of and-nodes are leaves that share no variables (the node is a *cube*).

- $f\text{-NNF}$: language of all flat $\text{NNF}$s;
- $\text{CNF}$: subset of $\text{NNF}$ satisfying simple disjunction;
- $\text{DNF}$: subset of $\text{NNF}$ satisfying simple conjunction.

- $\text{MODS}$: subset of $\text{DNF}$ where every formula satisfies determinism and smoothness (a formula in $\text{MODS}$ is represented by enumerating the set of its models).

- $\text{PI}$: formulas are in prime implicate form (conjunction of all prime implicates);
- $\text{IP}$: formulas are in prime implicant form (disjunction of all prime implicants).
5. Compact representation and compilation

**decision property** a decision node $N$ in an NNF $\varphi$ is a node labeled with TRUE, FALSE, or is an or-node having the form $(X \land \alpha) \lor (\neg X \land \beta)$, where $X$ is a variable, and $\alpha, \beta$ are decision nodes. In that case, $dVar(N)$ denotes the variable $X$.

- **BDD (binary decision diagrams)**: set of NNF formulas whose root is a decision node.

- **FBDD (free binary decision diagrams)** is the intersection of DNNF and BDD (on each path from the root to a leaf, a variable appears at most once).

**ordering** let $<$ be a total ordering on the variables. The language $\text{OBDD}_<$ is the subset of FBDD satisfying the following property: if $N$ and $M$ are or-nodes, and if $N$ is an ancestor of node $M$, then $dVar(N) < dVar(M)$.

- **OBDD (ordered binary decision diagrams)**: union of all $\text{OBDD}_<$ languages.
5. Compact representation and compilation

represents the binary decision diagram
5. Compact representation and compilation

Succinctness of compiled theories

MODS → OBDD < OBDD < NNF → DNNF → d-DNNF → sd-DNNF

DNF → FBDD → OBDD

IP → PI

CNF
An example of proof:

**Proposition:** CNF and DNF are not at least as succinct as OBDD.

**Sketch of proof:** $O_n = \bigoplus_{i=0}^{n} x_i$ parity function. $O_n$ has a linear-size OBDD representation but any CNF and any DNF representation of $O_n$ is exponentially large.
5. Compact representation and compilation

Another example of proof:

**Proposition:** DNNF is not at least as succinct as CNF, *unless the polynomial hierarchy collapses at the second level.*

**Sketch of proof:** consequence of these two lemmas:

**Lemma 1** (Cadoli and Donini, 97; Selman and Kautz, 96). If there exists a polysize compilation function $F$ mapping CNF formulas to propositional formulas such that (a) for any clause $C$ and for any CNF $\alpha$, $\alpha \models C$ iff $F(\alpha) \models C$, and (b) checking whether $F(\alpha) \models C$ is in P, then the polynomial hierarchy collapses at the second level.

**Lemma 2** (Darwiche 99): clausal entailment in DNNF is in P.
5. Compact representation and compilation

Complexity of querying a compiled theory

Complexity of the following key problems:

- checking whether $\varphi$ is satisfiable;
- checking whether $\varphi$ is valid;
- checking whether $\varphi \models C$, where $C$ is a clause;
- checking whether $\varphi$ and $\psi$ are equivalent;
- counting the models of $\varphi$. 
Polynomial-time satisfiability and clausal entailment:
- **YES** for **DNNF** and below;
- **NO** for **NNF** and **CNF**.
Polynomial-time equivalence check:
- YES for OBDD, IP, PI;
- NO for NNF, DNNF, DNF and CNF;
- open for d-DDNF, sd-DNNF and FBDD.
Polynomial-time model counting:

- YES for d-DNNF and below;
- NO for other classes.
5. Compact representation and compilation

**Complexity of transforming a compiled theory**

**conjunction** given formulas $\varphi_1, \ldots, \varphi_n$ of a given subset $C$ of $\mathbb{NNF}$, is it possible to compute in polytime a formula of $C$ equivalent to $\varphi_1 \land \ldots \land \varphi_n$?

**bounded conjunction** given two formulas $\varphi_1, \varphi_2$ of $C$, is it possible to compute in polytime a formula of $C$ equivalent to $\varphi_1 \land \varphi_2$?

**(bounded or not) disjunction** similar definitions

**forgetting** given $\varphi$ of $C$ and a subset $X$ of variables, is it possible to compute in polytime a formula of $C$ equivalent to $\exists X. \varphi$?

$\exists X. \varphi = \text{result of forgetting variables of } X \text{ in } \varphi = \text{projection of } \varphi \text{ on the language generated by } X$. 


Polynomial-time forgetting:
- YES for DNNF, DNF, PI and MODS;
- NO for FBDD, OBDD, OBDD< and IP;
- NO unless P = NP for NNF, d-DNNF, sd-DNNF and CNF.