

GEM 1501 Problem Solving With Computers

Lecture 7:

Inefficiency and Intractability

Frank Stephan

Previous Lecture

- 12.00-12.30h: Midterm Test
<http://www.comp.nus.edu.sg/~gem1501/index.html>
- Java Script Arrays
- Input and Output in Java Script
- Optimizing Speed

Moving Array Elements Around

Command	x	y
<code>var x = 0;</code>	0	-
<code>var y = new Array(1,2,3);</code>	0	(1,2,3)
<code>x = y[1];</code>	2	(1,2,3) << y[0],y[1],y[2]
<code>y.push(4,5,6,7,8);</code>	2	(1,2,3,4,5,6,7,8)
<code>y.push(9);</code>	2	(1,2,3,4,5,6,7,8,9)
<code>x = y.pop();</code>	9	(1,2,3,4,5,6,7,8)
<code>x = y.shift();</code>	1	(2,3,4,5,6,7,8)
<code>y.unshift(0,x);</code>	1	(0,1,2,3,4,5,6,7,8)
<code>x = y.slice(2,4);</code>	(2,3)	(0,1,2,3,4,5,6,7,8) << NOTE THE
<code>x = y.splice(2,4);</code>	(2,3,4,5)	(0,1,6,7,8) << DIFFERENCE
<code>var z = x.concat(y);</code>	x,y as above,	z is (2,3,4,5,0,1,6,7,8)

Input and Output in Java Script

- Usage of current window (document)
document.write(...); usage of html-commands
- Opening of Additional windows
- Preprogrammed windows

```
do { n = window.prompt("Input a value ",4);  
    k=1; for (m=1;m<=n;m=m+1) { k=k*m; }  
    window.alert("The faculty of "+n+" is "+k); }  
while (window.confirm("Press 'OK' for more computations"))
```

- Forms instead of Writing and Prompting
buttons and other environments in forms
The variable document.forms[i].elements[j].value

Today's Lecture

- Growth of functions
- NP-complete problems
- Is P equal NP?
- Even worse problems

Complexity of Towers of Hanoi

- For n rings and 3 pegs, the number of moves is $2^n - 1$.
- On a 1 GHz computer with one move per clock cycle, it would take 36 years to move 60 rings!
- 70 rings would take about 40000 years!
- 80 rings would take about 50 million years!

Monkey Puzzles

- Some tiles in an array, for example a 3 times 3 array on <http://www.comp.nus.edu.sg/~gem1501/monkeypuzzle.html>
- Interactive: Exchange tiles until neighbouring tiles have the same digits in those corners where they touch. Solution is sequence of moves.
- Puzzle: given a list of tiles, arrange them in the array such that neighbouring tiles have the same digits in those corners where they touch.

Example of Monkey Puzzles

m times n tiles, here an example of 3 times 5

12	14	21	22	35	38	41	43	48	54	54	75	88	89	92
61	22	12	11	89	75	76	38	21	47	92	24	12	54	41

Can they be arranged such that the numbers in corners are the same?

43	35	54	48	88
38	89	92	21	12

38	89	92	21	14
75	54	41	12	22

75	54	41	12	22
24	47	76	61	11

Naive Algorithm: Test all possibilities

This takes a long time: 130 76743 68000 possibilities.

More Info on Monkey Puzzle

- Assume an $m \times k$ board with tiles.
- Arrange $n = m \times k$ tiles so that the edges match.
- Simple solution: Try all possible ways!
- This approach is “impractical”! Better algorithms needed.
- There are n factorial many possibilities.
Note that $2^n < n! < n^n$ for $n \geq 4$.
- Next pages: $n!$ for $n = 32$ (4×8 board) and $n = 256$ (16×16 board).

Large Numbers, Powers of 32

Square of 32.

1024

Cube of 32.

32768

Fourth power of 32.

10 48576

32-th power of 2.

42949 67296

Factorial of 32.

2 63130 83693 36935 30167 21801 21600 00000

32-th power of 32.

1461 50163 73309 02918 20368 48327 16283 01965 59325 42976

<http://www.comp.nus.edu.sg/~gem1501/factorial.html>

Powers of 256

Square of 256.

65536

Cube of 256.

167 77216

Fourth power of 256.

42949 67296

256-th power of 2.

08687 90785 32699 84665 115 79208 92373 16195 42357 09850
64056 40394 57584 00791 31296 39936

Factorial of 256.

85 78177
75342 84265 41190 82271 68123 26251 57781 52027 94856 19859
65565 03772 69452 55314 75893 77440 29136 04514 08450 37588
53423 36584 30615 71968 34693 69647 53222 89288 49742 60256
79637 33256 33687 86442 67520 76267 94560 18796 88679 71521
14330 77020 77526 64645 14647 09187 32610 08328 76325 70281
89807 73671 78145 41702 50523 01860 84953 19068 13825 74810
70252 81755 94594 76987 03466 57127 38139 28620 52347 56808
21886 07012 03611 08315 20935 01947 43710 91017 26968 26286
16062 63662 43502 28409 44191 40842 46159 36000 00000 00000
00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

Intractable Problems

- Polynomial problems can be solved in time $O(n^k)$ for some constant k , for example $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, $O(n^4)$, $O(n^5)$.
- Exponential problems: $O(2^{\sqrt{n}})$, $O(1.333^n)$, $O(2^n)$, $O(3^n)$, $O(4^n)$, $O(n!)$, $O(n^n)$, $O(2^{n^2})$, $O(n^2!)$.
- 10^{10} steps can easily be done, 10^{20} cannot be done.

Category	Steps	Tractable	Clearly Intractable
	$f(n)$	n with $f(n) \leq 10^{10}$	n with $f(n) \geq 10^{20}$
Polynomial	n^2	100000	100000000000
	n^5	100	10000
Exponential	$2^{\sqrt{n}}$	1600	6400
	1.333^n	70	160
	2^n	40	80
	$n!$	15	22
	n^n	10	16
	$n^2!$	3	5
Double Expo.	2^{2^n}	5	7

Can we wait for faster computers?

- What if we use a computer 100 times faster?
What if we use 1000 computers hooked together?
- 1000^4 and 2^{50} operations are doable, $50!$ and 2^{100} not.
- If size m is solvable and speed increased 100 times:
This solves size $10 * m$ problems for $O(n^2)$ algorithm;
This solves size $m + 6$ problems for $O(2^n)$ algorithm.
- Polynomial problems and exponential problems really behave differently!

Summary: Run-Time of Algorithms

- Let n be the length of the input (number of bits or number of symbols);
- Let $T(n)$ be the time needed by the algorithm for the most difficult input consisting of up to n symbols;
- Two major types of algorithms (among others):
Algorithm runs in polynomial time iff $T(n) \in O(n^c)$ for some constant c ;
Algorithm runs in exponential time iff $T(n) \in O(2^{n^c})$ for some constant c .

Problems and Solutions

- A decision problem is a set A of instances x such that $x \in A$ iff x has a solution y .
- Example: Monkey-Puzzles would be the set of all $(m, k, a_1, \dots, a_{m \cdot k})$ such that the tiles $a_1, \dots, a_{m \cdot k}$ can be arranged in an $m \times k$ array.
- A problem A is in P iff some algorithm can compute for every x in polynomial time whether $x \in A$ or $x \notin A$, that is, iff some algorithm can find out in polynomial time whether an instance x has a solution or not.
- A problem A is in NP iff some algorithm can verify in polynomial time whether a given solution y for a given instance x is correct; size of solutions should be bounded by a polynomial in size of x .
- A problem A is NP-complete iff A is in NP and for every further problem B in NP there is a polynomial time computable function F with $x \in B \Leftrightarrow F(x) \in A$ for all x .

Computing and Verifying Solutions

Mathematical Task: Find numbers $a, b, c, d, e, f, g, h, i, j, k, l \in \{-1, +1\}$ such that

$$(a + 2b + 3c + 4d + 5e + 6f + 7g + 8h + 9i + 10j + 11k + 12l)^2 + (a + b + d + e)^2 + (i + j + k + l)^2 + (a + c + d + e + 3g + 3h)^2 = 0$$

for these numbers; that is, compute a solution to the equation making the above polynomial 0.

Can be done but needs some time.

Solution is $a = +1, b = -1, c = -1, d = +1, e = -1, f = +1, g = +1, h = -1, i = -1, j = +1, k = +1, l = -1$: this solution makes all squares 0 and hence the whole sum 0.

Verifying solutions is for humans easier than finding them. Is the same true for computers?

Zeroes of Multivariate Polynomials

- Mathematics: Has a function a 0 on some set?
Limiting complexity: Considering only quadratic multivariate polynomials and only places where variables are -1 or 1 .

- Let QMP contain all functions of the form

$$p(x_1, x_2, \dots, x_n) = \sum_{i,j} a_{i,j} \cdot x_i \cdot x_j.$$

where all $a_{i,j} \in \{n, -n + 1, \dots, -2, -1, 0, 1, 2, \dots, n - 1, n\}$.

An example of a function in QMP is

$$p(x_1, x_2, x_3) = x_1 \cdot x_2 - x_2 \cdot x_3$$

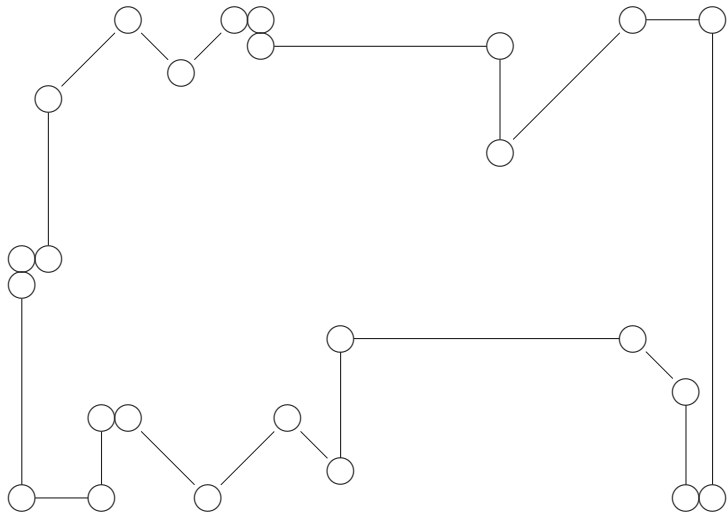
where $a_{1,2} = 1, a_{2,3} = -1$ and all other $a_{i,j} = 0$.

- Question: Is $p(x_1, x_2, \dots, x_n) = 0$ for some $x_1, x_2, \dots, x_n \in \{-1, 1\}$?
- The set $\{p \in \text{QMP} : \exists x_1, x_2, \dots, x_n \in \{-1, 1\} (p(x_1, \dots, x_n) = 0)\}$ is an NP-complete problem.

Travelling Salesman Problem

A salesman wants to visit from his hometown 24 other places for doing business and then return home.

He does not visit any place twice.



NP-Formulation

Given the cities
and a constant c ,
is there a tour
of length up to c ?

Shortest tours can be found for several 10000 cities.

Colourability

- The Four Colour Theorem provides an algorithm to colour a map of connected countries with four colours such that no neighbouring countries have the same colour.
- For some maps, this can be done with three colours.
- Given a map as a graph (countries involved, pairs of countries which are neighbouring to each other), find out whether it is possible to assign to the countries colours "white", "red" and "blue" such that for no pair of countries, both have the same colour.
- The type of graphs which come from maps are called "planar".
- It is NP-complete to decide which planar graphs can be coloured with 3 colours.

Maps and Graphs

Map

```
www bbbbbbb  
www bbbbbbb  
  ww bbbbb  
bb rrr ww  
bb rr www
```

A map of five countries coloured in white (w), blue (b) and red (r), respectively.

Planar Graph

```
  w - b  
 / \ / \  
b - r - w
```

A graph with nodes in the three colours, edges connect neighbouring countries which should have different colours.

"Planar" = "Graph drawable in plane"

Satisfiability

- Let $\phi = x_1 \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_4 \vee (x_5 \wedge \neg x_6)) \wedge (\neg x_2 \vee x_5)$ be a Boolean formula over Boolean variables.
- An assignment defines to every variable a truth-value; a common short hand is 0 for false and 1 for true.
- The assignment $(1, 0, 1, 0, 0, 0)$ satisfies ϕ ; it evaluates to 1:
 $1 \wedge (0 \vee 1) \wedge (\neg 0 \vee 0 \vee (0 \wedge \neg 0)) \wedge (\neg 0 \vee 0)$ is $1 \wedge 1 \wedge (1 \vee 0 \vee (0 \wedge 1)) \wedge (1 \vee 0)$
which is the conjunction of four times a 1.
- The assignment $(0, 1, 0, 1, 0, 1)$ does not satisfy ϕ ; it evaluates to 0:
 $0 \wedge (1 \vee 0) \wedge (\neg 1 \vee 1 \vee (0 \wedge \neg 1)) \wedge (\neg 1 \vee 0)$.
- A formula is satisfiable iff there is a satisfying assignment of its variables. So the above formula is satisfiable and $x_1 \wedge x_2 \wedge (\neg x_1 \vee \neg x_2)$ is not.

Seen so far NP-Complete Problems

- In P: Quadratic Equation in one variable.
- NP-complete: Multivariate Quadratic Equations.
Number of variables varies from instance to instance, variables take fixed range of values.
- NP-complete: Travelling Salesman Problem.
- NP-complete: Monkey Puzzles.
- NP-complete: 3-Colourability of planar graphs.
- NP-complete: Satisfiability (SAT), the set of all formulas ψ (in conjunctive normal form) which be made true by suitable choice of variables.
- Next slides: more on satisfiability.

3SAT

- 3SAT (special variant)
- n variables and a set of up to $8n^3$ disjunctions (called clauses) of up to 3 variables and negated variables, for example, $x_1 \vee x_2 \vee x_3$, $x_4 \vee \neg x_5$, $\neg x_2 \vee \neg x_5$, $x_3 \vee x_4$.
- Is it possible to assign truth-values to the variables such that all clauses are true?
- Satisfiability has an $O(2^n)$ algorithm and 3SAT has an $O(1.333^n)$ algorithm. Better algorithms might exist.

A Positive Example

- The system of equations:

$$Cl_1: x_1 \vee x_2 \vee \neg x_5;$$

$$Cl_2: x_1 \vee x_2 \vee x_3;$$

$$Cl_3: \neg x_1 \vee x_4;$$

$$Cl_4: \neg x_1 \vee \neg x_4;$$

$$Cl_5: \neg x_2;$$

Full formula: $Cl_1 \wedge Cl_2 \wedge Cl_3 \wedge Cl_4 \wedge Cl_5$.

- This formula is satisfiable:

Take x_5 to be false to satisfy clause Cl_1 ;

Take x_3 to be true to satisfy clause Cl_2 ;

Take x_1 to be false to satisfy clauses Cl_3 and Cl_4 ;

Take x_2 to be false to satisfy clause Cl_5 ;

The variable x_4 can be either true or false (does not matter).

- Solutions are not always so easy to find.

A Negative Example

- The system of equations:

$$Cl_1: x_1 \vee x_2 \vee \neg x_3;$$

$$Cl_2: x_1 \vee x_2 \vee x_3;$$

$$Cl_3: \neg x_1 \vee x_4;$$

$$Cl_4: \neg x_1 \vee \neg x_4;$$

$$Cl_5: \neg x_2;$$

Full formula: $Cl_1 \wedge Cl_2 \wedge Cl_3 \wedge Cl_4 \wedge Cl_5$.

- This formula is not satisfiable:

x_2 must be false since otherwise clause Cl_5 is false;

x_1 must be false since otherwise one of the clauses Cl_3 and Cl_4 is false by case distinction over x_4 ;

x_3 must be true since otherwise clause Cl_2 is false;

All literals are false in Cl_1 and Cl_1 cannot be made true.

- Such derivations of unsatisfiability can have exponential length, as one might have conditions involving several variables instead of only one.

Resolution: Algorithm to check Satisfiability

For each variable y do the case which applies:

- If neither y nor $\neg y$ occur in any clause then this step is void.
- If y but not $\neg y$ occurs in clauses then remove all clauses in which y appears.
- If $\neg y$ but not y occurs in clauses then remove all clauses in which $\neg y$ appears.
- If y and $\neg y$ both occur as clauses consisting of 1 literal only then return with output “Instance is not satisfiable.”
- Otherwise, for each clause of the form $y \vee \psi$ and each clause of the form $\neg y \vee \phi$, if $\psi \vee \phi$ does not contain $z \vee \neg z$ for any z then add $\psi \vee \phi$ to the set of all clauses.
Having done this, remove all clauses containing y or $\neg y$ from the set of all clauses.

If the algorithm has not terminated within the loop then return with the output “Instance is satisfiable.”

Resolution Example 1

- Given $x_1 \vee x_2 \vee \neg x_5$; $x_1 \vee x_2 \vee x_3$; $\neg x_1 \vee x_4$; $\neg x_1 \vee \neg x_4$; $\neg x_2$
- Resolving x_1 gives:
 $x_2 \vee x_4 \vee \neg x_5$; $x_2 \vee \neg x_4 \vee \neg x_5$; $x_2 \vee x_3 \vee x_4$; $x_2 \vee x_3 \vee \neg x_4$; $\neg x_2$
- Resolving x_2 gives:
 $x_4 \vee \neg x_5$; $\neg x_4 \vee \neg x_5$; $x_3 \vee x_4$; $x_3 \vee \neg x_4$
- Assume x_3 to be true gives:
 $x_4 \vee \neg x_5$; $\neg x_4 \vee \neg x_5$
- Resolving x_4 gives:
 $\neg x_5$
- Assume x_5 to be false gives a void instance.
So the original one is satisfiable.

Resolution Example 2

- Given $x_1 \vee x_2 \vee \neg x_3$; $x_1 \vee x_2 \vee x_3$; $\neg x_1 \vee x_4$; $\neg x_1 \vee \neg x_4$; $\neg x_2$
- Resolving x_1 gives:
 $x_2 \vee \neg x_3 \vee x_4$; $x_2 \vee \neg x_3 \vee \neg x_4$; $x_2 \vee x_3 \vee x_4$; $x_2 \vee x_3 \vee \neg x_4$; $\neg x_2$
- Resolving x_2 gives:
 $\neg x_3 \vee x_4$; $\neg x_3 \vee \neg x_4$; $x_3 \vee x_4$; $x_3 \vee \neg x_4$
- Resolving x_3 gives:
 $x_4 \vee x_4$; $\neg x_4 \vee \neg x_4$; $x_4 \vee \neg x_4$ (ignored)
- Not Satisfiable as x_4 and $\neg x_4$ both exist

Evaluation

- In the case of 2SAT, Resolution runs in polynomial time as whenever $\psi \vee \phi$ is introduced while resolving a variable, both parts contain at most one literal and so all clauses remain to have at most 2 variables. 2SAT is polynomial time computable.
- In the case of 3SAT, the number of literals per clause can grow and the number of clauses can become exponential. Resolution needs exponential time on some 3SAT instances.

What does “NP” stand for?

- P stands for the “polynomial time computable problems.”
- NP stands for “non-deterministic polynomial computable problems.”
- A “non-deterministic computer” is a computer that guesses a solution before it says “This equation has a solution” but it has no proof when saying “This equation does not have a solution.”
- The basic phenomenon of NP problems is known from mathematics: Given a function f , it might be difficult to know whether there is an x at which $f(x)$ is 0. But given such an x , one can easily verify that $f(x)$ is really 0.
- “Intuition” and “experience” might tell an expert how to solve an NP-type problem. Before handing out the solution to others, an easy verification is done to make sure that the solution is correct.

Why Only Decision Problems

- Let A_0 be a solvable 3SAT instance. For $m = 1, 2, \dots, n$:
 - Test whether $A_{m-1} \cup \{x_m\}$ or $A_{m-1} \cup \{\neg x_m\}$ is solvable.
 - Let $A_m = A_{m-1} \cup \{x_m\}$ if that is solvable and $A_m = A_{m-1} \cup \{\neg x_m\}$ otherwise.
- Now A_n has a unique solution where x_m is set to be true iff the clause x_m is contained in A_n and x_m is set to be false iff the clause $\neg x_m$ is contained in A_n .
- The unique solution of A_n is fast to find and is a solution of A_0 .
- If 3SAT is in P one can compute a solution in polynomial time.
- In theoretical investigations, one just wants to know whether certain 3SAT instances have solutions. Every related information can be computed from the 3SAT decision problem fast.
- This is typical for all NP-complete decision problems.

NP-complete Problems Stand and Fall Together

- All NP-complete problems are equivalent!
- One NP-complete problem is in $P \Leftrightarrow$ all NP-problems are in P .
- One problem solvable in $O(n^k) \Rightarrow$ for every problem there is a constant c such that has an $O(n^c)$ algorithm, c can be very large.
- One problem requires $O(2^n) \Rightarrow$ for every problem there is a c such that it requires at least $O(2^{n^c})$ steps, c can be a very small constant.
- Reason for equivalence: NP-complete problems are transformable into each other: Given two NP-complete problems A, B there is a polynomial time function f with $x \in A \Leftrightarrow f(x) \in B$. If B is in P , so is A as one can take x , compute in polynomial time $f(x)$ and then check in polynomial time whether $f(x) \in B$.

Is P Equal to NP?

- Question lies at the heart of “Algorithmics”
- Unsolved since it was posed in 1971
- Many people believe that $P \neq NP$, but no one knows for sure.
- Mathematics: Is finding of a solution harder than verifying it?

Intermediate Problems

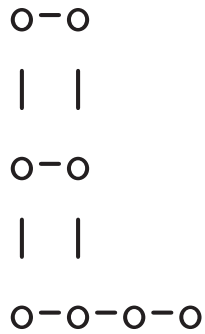
- If $P \neq NP$ then there are intermediate problems which are neither in P nor NP -complete. They are artificially constructed, but are there also natural ones? A natural candidate was the set of prime numbers.
- In 2002, Prof. Manindra Agrawal (IIT Kanpur) and two of his students, Nitin Saxena and Neeraj Kayal, gave a polynomial time algorithm. The test whether x is prime needs $O((\log x)^6 f(\log \log x))$ where f is a polynomial. If n is the number of decimal digits of x then the complexity is $O(n^6 \log(n))$.

Another Candidate

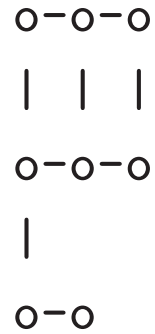
- A graph is a set of vertices together with a set of edges, each edge connecting two vertices.
- Consider the following two undirected graphs:
 $V_1 = \{1, 2, 3, 4\}$, $E_1 = \{(1, 2), (2, 3), (3, 4), (1, 3)\}$;
 $V_2 = \{1, 2, 3, 4\}$, $E_2 = \{(1, 2), (2, 3), (3, 4), (2, 4)\}$.
- These graphs are isomorphic:
The nodes i, j in V_1 are connected iff $5 - i, 5 - j$ are connected in V_2 . The mapping $i \rightarrow 5 - i$ verifies that one can obtain V_2 from V_1 by renaming the nodes. Such a mapping is called an isomorphism and graphs are isomorphic whenever an isomorphism exists.
- The problem whether two graphs are isomorphic is in NP: one can guess the isomorphism as a table of which vertex in V_1 corresponds to which vertex in V_2 and then verify that this table is indeed an isomorphism. Some people believe that it is neither in P nor NP-complete.

Example of Graph-Isomorphism

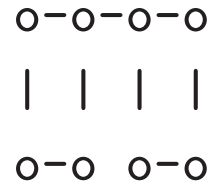
Graph 1



Graph 2



Graph 3



Graph 1 and Graph 2 are isomorphic.

Graph 3 is not isomorphic to any of these two.

Graph 1 and Graph 2 have a vertex with only edge,
in Graph 3 every node is linked to other nodes
via at least 2 edges.

More difficult problems

- Let ϕ be a formula with variables x_1, y_2, \dots, x_n and y_1, y_2, \dots, y_n .
- Consider the following problems:
 1. What is $\exists x_1 \exists x_2 \dots \exists x_n \forall y_1 \forall y_2 \dots \forall y_n \phi$?
 2. What is $\exists x_1 \forall y_1 \exists x_2 \forall y_2 \dots \exists x_n \forall y_n \phi$?
- All these problems can be computed by testing out all possible values for the variables in suitable order. Such an algorithm needs exponential time as it analyzes 4^n possible assignments, but its variables need only polynomial space. The complexity class PSPACE captures the decision problems which can be solved by such algorithms.
- The second problem is complete for PSPACE. The first one is in a subclass of PSPACE which is called NP^{NP} .
- $P \subseteq NP \subseteq \text{NP}^{\text{NP}} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$ and $P \subset \text{EXPTIME}$. More is not known.

Checkers

At checkers, two players — Anke (A) and Boris (B) — play on an 8×8 board moving their pieces trying to capture or block the pieces of an opponent. Here the pieces are marked with their initials.

	(B)		(B)		(B)		(B)
(B)		(B)		(B)		(B)	
	(B)		(B)		(B)		(B)
...		
	(A)	
...		(A)		(A)		(A)	
	(A)		(A)		(A)		(A)
(A)		(A)		(A)		(A)	

See also on <http://www.darkfish.com/checkers/Checkers.html>

How difficult is it for Anke to play as good as optimal? That is, how difficult is it for her not to make any mistake, with whomever she plays. A winning strategy for Anke gives in every situation the best possible move.

Games and Decision Problems

- The class EXPTIME captures all decision problems which can be solved in exponential time.
- A winning strategy for checkers on an $n \times n$ board is EXPTIME-complete: More precisely the decision problem “Given situation X of Checkers and move Y , is this move optimal (that is, not a mistake)” is EXPTIME complete.
- EXPTIME-completeness needs that the game can run in exponential time; there are no polynomial time algorithms for this problem.
- If a game ends after $m = p(n)$ moves for a polynomial p and board size n , then one can use the formula

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2 \dots \exists x_m \forall y_m (\text{Player wins from } a \text{ with these moves})$$

to check whether the player can win from a given situation a . This problem is in PSPACE. For example, making the best move in $n \times n$ Reversi (also known as Othello) is PSPACE-complete.

- See <http://www.ics.uci.edu/~eppstein/cgt/hard.html> for more information on hard games.

Worse Than Exponential

- There are problems that need a runtime of 2^{2^n} .
- Example: Presburger arithmetic
Set of all true quantified formulas over integer numbers using addition, subtraction and comparisons like $\exists x \forall y (x + x + x + y + y \neq 0)$ and $\forall x \exists y (y > x + 5)$.
- These problems are called “double exponential”.
- There are also “triple exponential” and “four times exponential” problems and so on.
- Beyond that: “nonelementary problems”.

Overview of Today's Lecture

- Growth of functions
- NP-complete problems
- Is P equal NP ?
- Even worse problems

Next Week - The Undecidable

- Certain things are not only slow but just impossible
- Unbounded Puzzles, Post's Correspondence Problem
- The Halting-Problem
- Recursively Enumerable Sets
- Rice's Theorem