CS481: Bioinformatics Algorithms

Can Alkan
EA509
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/

CS481

- Class hours:
  - Mon 10:40 - 12:30; Thu 9:40 - 10:30
- Class room: EB201
- Office hour: Tue + Thu 11:00-12:00
- TA: Fatma Kahveci (fatmaba@gmail.com)
- Grading:
  - 1 midterm: 25%
  - 1 final: 35%
  - Homeworks (theoretical & programming [C/C++]): 20%
  - Quizzes: 20%

CS481

- Textbook: An Introduction to Bioinformatics Algorithms (Computational Molecular Biology), Neil Jones and Pavel Pevzner, MIT Press, 2004
- Recommended Material
  - Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison, Cambridge University Press
  - Bioinformatics: The Machine Learning Approach, Second Edition, Pierre Baldi, Soren Brunak, MIT Press
  - Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Dan Gusfield, Cambridge University Press
- (Most) of the course material is publicly available at: www.bioalgorithms.info

CS481

- This course is about **algorithms** in the field of bioinformatics:
  - What are the problems?
  - What algorithms are developed for what problem?
  - Algorithm design techniques
- This course is **not** about how to analyze biological data using available tools:
  - Recommended course: MBG 326: Introduction to Bioinformatics

CS481 and other courses

- Includes elements from:
  - CS201/202: data structures
  - CS473: algorithms, dynamic programming, greedy algorithms, branch-and-bound, etc.
  - CS476: complexity, context-free grammars, DFA/NFA
  - CS464: hidden Markov models

CS481: Assumptions

- You are assumed to know/understand
  - Computer science basics (CS101/102 or CS111/112)
    - CS201/202 would be better
    - CS473 would be even better
  - Data structures (trees, linked lists, queues, etc.)
  - Elementary algorithms (sorting, hashing, etc.)
  - Programming: C, C++
- You don’t *have to* be a “biology expert”: MBG 110 would be sufficient

Bioinformatics

- Development of methods based on computer science for problems in biology and medicine
  - Sequence analysis (combinatorial and statistical/probabilistic methods)
  - Graph theory
  - Data mining
  - Database
  - Statistics
  - Image processing
  - Visualization
  - …..

CS 481

Bioinformatics: Applications

- Human disease
  - Personalized Medicine
- Genomics: Genome analysis, gene discovery, regulatory elements, etc.
- Population genomics
- Evolutionary biology
- Proteomics: analysis of proteins, protein pathways, interactions
- Transcriptomics: analysis of the transcriptome (RNA sequences)
- …

Why would you learn these algorithms?

- Most developed for research within other fields that include string processing, clustering, text-pattern search, etc.
- Bioinformatics (non-academic) jobs on the rise:
  - Genomics England, Genome Asia, etc.: 100,000 genome projects
  - DNAnexus, SevenBridges: genome analysis on the cloud.
    - SevenBridges has Ankara and Istanbul offices

(VERY) BRIEF INTRODUCTION TO COMPLEXITY

Tractable vs intractable

- Tractable algorithms: there exists a solution with O(f(n)) run time, where f(n) is *polynomial*
- *P* is the set of problems that are **known** to be *solvable* in polynomial time
- *NP* is the set of problems that are *verifiable* in polynomial time
  - *NP: "non-deterministic polynomial"*

$$P \subset NP$$

NP-hard

- *NP-hard:* non-deterministic polynomial hard
  - Set of problems that are "*at least as hard as the hardest problems in NP*"
  - There are no known polynomial time optimal solutions
  - There *may* be polynomial-time *approximate* solutions

NP-Complete

- A *decision problem* C is in NPC if :
  - C is in NP
  - Every problem in NP is **reducible** to C in polynomial time

That means: if you could solve any NPC problem in polynomial time, then you can solve all of them in polynomial time

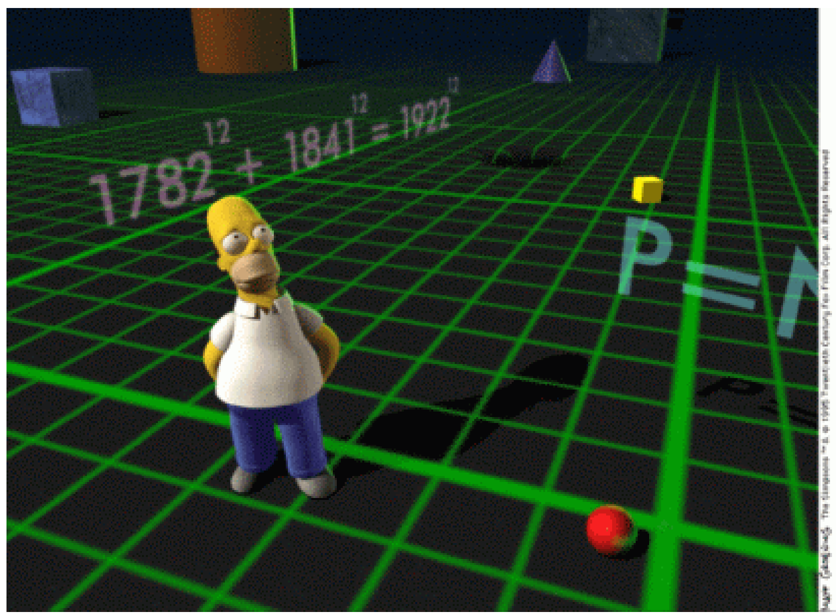
Decision problems: outputs “yes” or “no”

NP-intermediate

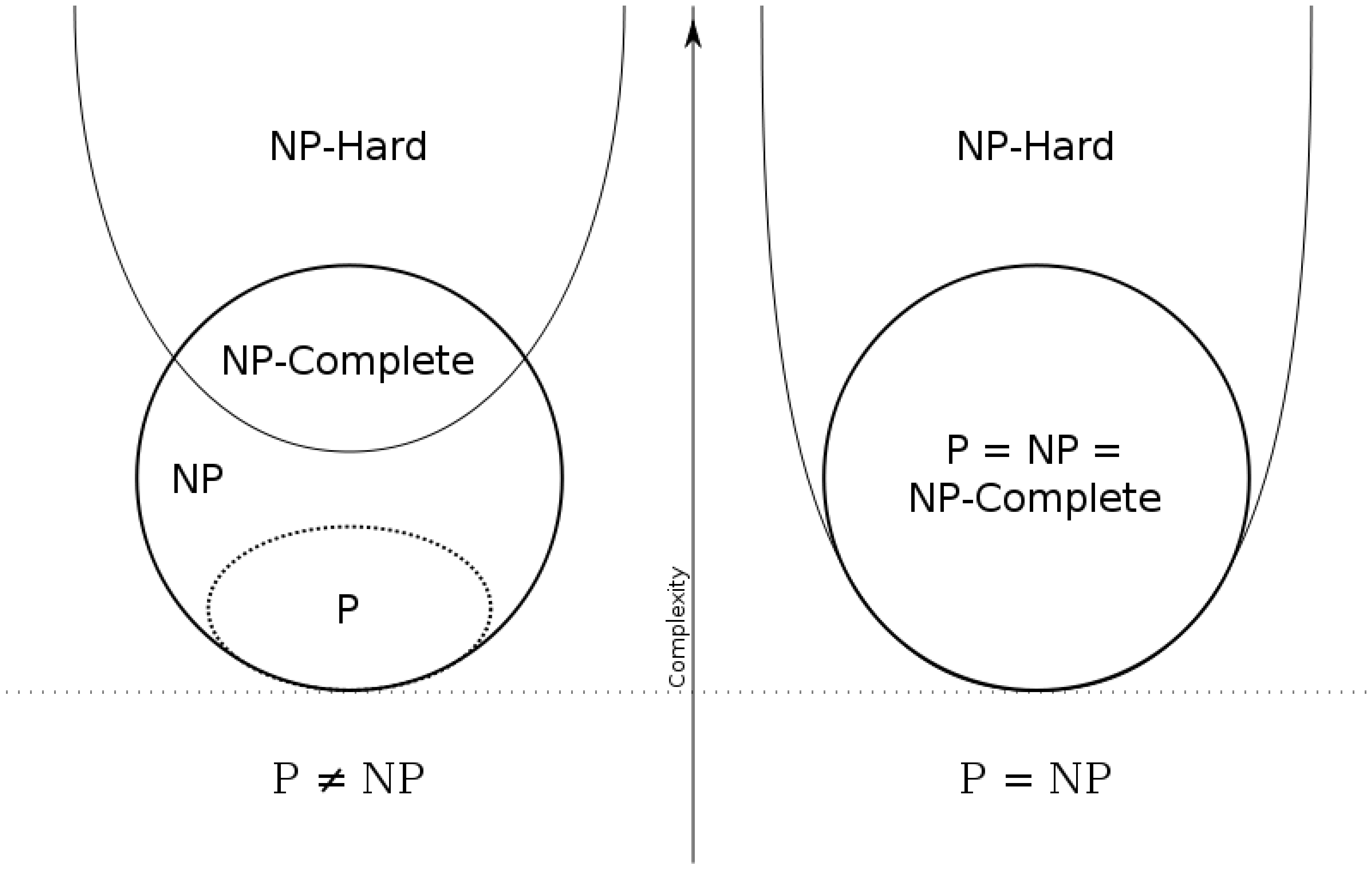
- Problems that are in NP; but not in either NPC or NP-hard

P vs. NP

- We do not know whether P=NP or P≠NP
  - Principal unsolved problem in computer science
  - It is believed that P≠NP



P vs. NP vs. NPC vs. NP-hard



Examples

- P:
  - Sorting numbers, searching numbers, pairwise sequence alignment, etc.
- NP-complete:
  - Subset-sum, traveling salesman, etc.
- NP-intermediate:
  - Factorization, graph isomorphism, etc.

Historical reference

- The notion of NP-Completeness: Stephen Cook and Leonid Levin independently in 1971
  - First NP-Complete problem to be identified: Boolean satisfiability problem (SAT)
    - Cook-Levin theorem
- More NPC problems: Richard Karp, 1972
  - “21 NPC Problems”
- Now there are thousands….

ALGORITHM DESIGN TECHNIQUES

Sample problem: Change

- Input: An amount of money M, in cents
- Output: Smallest number of coins that adds up to M
  - Quarters (25c): q
  - Dimes (10c): d
  - Nickels (5c): n
  - Pennies (1c): p
  - Or, in general, $c_1, c_2, \ldots, c_d$ (*d* possible denominations)

Algorithm design techniques

- **Exhaustive search / brute force**
  - Examine every possible alternative to find a solution

BruteForceChange$(M, \mathbf{c}, d)$

$smallestNumberOfCoins \leftarrow \infty$
for each $(i_1, \ldots, i_d)$ **from** $(0, \ldots, 0)$ **to** $(M/c_1, \ldots, M/c_d)$
    $valueOfCoins \leftarrow \sum_{k=1}^{d} i_k c_k$
    if $valueOfCoins = M$
        $numberOfCoins \leftarrow \sum_{k=1}^{d} i_k$
        if $numberOfCoins < smallestNumberOfCoins$
            $smallestNumberOfCoins \leftarrow numberOfCoins$
            bestChange $\leftarrow (i_1, i_2, \ldots, i_d)$
return (**bestChange**)

Algorithm design techniques

- **Greedy algorithms:**
  - Choose the “most attractive” alternative at each iteration

$\textsc{BetterChange}(M, \mathbf{c}, d)$

$r \leftarrow M$
for $k \leftarrow 1$ **to** d
  $i_k \leftarrow r / c_k$
  $r \leftarrow r - c_k \cdot i_k$
return $(i_1, i_2, \ldots, i_d)$

$\textsc{USChange}(M)$

$r \leftarrow M$
$q \leftarrow r / 25$
$r \leftarrow r - 25 \cdot q$
$d \leftarrow r / 10$
$r \leftarrow r - 10 \cdot d$
$n \leftarrow r / 5$
$r \leftarrow r - 5 \cdot n$
$p \leftarrow r$
return (q, d, n, p)

Algorithm design techniques

- **Dynamic Programming:**
  - Break problems into subproblems; solve subproblems; merge solutions of subproblems to solve the real problem
  - Keep track of computations to avoid recomputing values that you already solved
  - *Dynamic programming table*

DP example: Rocks game

- Two players
- Two piles of rocks with p_1 rocks in pile 1, and p_2 rocks in pile 2
- In turn, each player picks:
  - One rock from either pile 1 or pile 2; OR
  - One rock from pile 1 and one rock from pile2
- The player that picks the last rock wins

DP algorithm for Player 1

- Problem: $p_1 = p_2 = 10$
- Solve more general problem of $p_1 = n$ and $p_2 = m$
- It's hard to directly calculate for n=5 and m=6; we need to solve smaller problems

DP algorithm for Player 1

pile1 \ pile2	0	1	2	3	4	5	6	7	8	9	10
0		W									
1	W	W									
2											
3											
4											
5											
6											
7											
8											
9											
10											

Initialize; obvious win for Player 1 for 1,0; 0,1 and 1,1

DP algorithm for Player 1

pile1 \ pile2	0	1	2	3	4	5	6	7	8	9	10
0		W	L								
1	W	W									
2	L										
3											
4											
5											
6											
7											
8											
9											
10											

Player 1 cannot win for 2,0 and 0,2

DP algorithm for Player 1

pile2

pile1		0	1	2	3	4	5	6	7	8	9	10
	0		W	L								
pile1	1	W	W	W								
	2	L	W									
	3											
	4											
	5											
	6											
	7											
	8											
	9											
	10											

Player 1 can win for 2,1 if he picks one from pile2

Player 1 can win for 1,2 if he picks one from pile1

DP algorithm for Player 1

pile2

pile1

Player 1 can win for 2,1 if he picks one from pile2

Player 1 can win for 1,2 if he picks one from pile1

DP algorithm for Player 1

	pile2	0	1	2	3	4	5	6	7	8	9	10
	0		W	L								
pile1	1	W	W	W								
	2	L	W	L								
	3											
	4											
	5											
	6											
	7											
	8											
	9											
	10											

Player 1 cannot win for 2,2

Any move causes his opponent to go to W state

DP “moves”

When you are at position *(i,j)*

Go to:

Pick from pile 1: ***(i-1, j)***

Pick from pile 2: ***(i, j-1)***

Pick from both piles 1 and 2: ***(i-1, j-1)***

DP final table

	0	1	2	3	4	5	6	7	8	9	10
0		W	L	W	L	W	L	W	L	W	L
1	W	W	W	W	W	W	W	W	W	W	W
2	L	W	L	W	L	W	L	W	L	W	L
3	W	W	W	W	W	W	W	W	W	W	W
4	L	W	L	W	L	W	L	W	L	W	L
5	W	W	W	W	W	W	W	W	W	W	W
6	L	W	L	W	L	W	L	W	L	W	L
7	W	W	W	W	W	W	W	W	W	W	W
8	L	W	L	W	L	W	L	W	L	W	L
9	W	W	W	W	W	W	W	W	W	W	W
10	L	W	L	W	L	W	L	W	L	W	L

Also keep track of the choices you need to make to achieve W and L states: ***traceback table***

Algorithm design techniques

- **Divide and conquer:**
  - Split, solve, merge
    - Mergesort
- **Machine learning:**
  - Analyze previously available solutions, calculate statistics, apply most likely solution
- **Randomized algorithms:**
  - Pick a solution randomly, test if it works. If not, pick another random solution