Programming Projects

(Last updated: January 31, 2014)

Teams:

You are to work in teams of two students. Each team will work on a project, do the design, implementation, testing, and prepare a final report. Both members of the team are expected to participate equally in all aspects of the project. Both members of the team will receive the same grade on the project. You are expected to form groups and choose a topic by Thursday, February 27, 2014.

Project Requirements:

The team is expected to do a project that uses or develops techniques, theory, or algorithms introduced in the class. Students are encouraged to develop projects that have some relationship to their own interests, but a list of suggested projects is presented to help get started. Some are general, others specific. You can

- Pick a project from the list,
- Modify a project to suit your interests, or
- Suggest your own project.

The key idea is to be creative either in developing a new algorithm or in implementing an existing one. Results, whether good or bad, should be compared with those obtained from existing implementations.

The project implementation may be developed for any platform. You can use C, C++, C#, Java, Matlab, Python, Ruby, or Perl. The World Wide Web contains many programs as well as the source code. You may use and modify such code provided appropriate acknowledgements and citations are made.

The final project grade will be computed in the following way:

a) Team Formation: 10% b) Progress Report: 20% c) Project report: 70%

Important Dates:

a) Team Formation

Thursday, February 27: Each team submits a paper containing the names of the team members and the chosen topic. If the topic is not the one described under "List of Suggested Projects" below, then the team includes a detailed description of their proposed project.

b) Progress Report

Thursday, April 10: Two-page progress report by each team is due at the beginning of the lecture. Include the title of the project and a description of what has been achieved. Alternatively, the team can submit all the work that has been done up to April 10, 2014.

c) Project Report

Tuesday, May 6: Printed copies of the final project are due at the beginning of the lecture. Do not forget to include all pertinent documents. Also, submit a CD or USB flash drive containing all the files described under "Submission Requirements" below.

Submission Requirements:

A) Submit a hard copy of the project containing all of the following:

1) Title page.

2) Table of contents (with page numbers).

3) An essay of not more than 6, and not less than 4, 1 and 1/2 spaced pages (font size: 11 or 12 points) describing the problem, the overall organization, design of your program, and results of your project. The essay should give the user an overall roadmap of your code. Include a flowchart, or UML or a structure chart to show the design of your program. Do not forget to number pages!

4) A one page description of the choice of the test data and the testing strategy you used.

5) Include one sample output of your program. Make sure that your output is readable and well formatted.

6) Instructions for running your program.

A good project report will include the following:

- Background of the problem from the literature search
- A clear definition of the problem
- An explanation and justification of methods of data analysis.
- A description and justification of the data sources.
- Analysis of the results and comparison with existing tools.
- Conclusions based on the results.
- Possible directions for future research.
- Instructions on how to compile and execute your program
- A full list of references.

B) Submit a CD, or USB flash drive, labeled with your names, course number and semester, and containing:

a) The source code (fully documented)
b) The input files
c) The document specified in part A

Please make sure that all the files on the CD or USB key are readable.

List of Suggested Projects

1. Problems from CLRS

Pick any problem and choose an algorithm to implement and test. A few suggestions:

- Rod cutting. Section 15.1, pages 360-370.
- Matrix-chain multiplication. Section 15.2, pages 370-378.
- Longest common subsequence. Section 15.4, pages 390-397.
- Edit distance. Problem 15-5, pages 406-408.
- Viterbi algorithm. Problem 15-7, pages 408-409.
- Scheduling to minimize average completion time. Prob 16-2, p 447.
- Off-line caching. Problem 16-5, page 449.
- Toeplitz matrices. Problem 30-2, page 921.
- Multidimensional fast Fourier transform. Prob 30-3, pages 921-922.

[CLRS] Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest, and C. Stein. MIT Press, 2009.

2. NP-Hard problems from CLRS

Pick any NP-hard problem and choose a heuristic algorithm or approximation algorithm to implement and test. A few suggestions:

- Clique problem. Section 34.5.1, pages 1086-1089.
- Vertex cover problem. Section 34.5.2, pages 1089-1091.
- Hamiltonian cycle problem. Section 34.5.3, pages 1091-1096.
- Traveling salesperson problem. Section 34.5.4, pages 1096-1097.
- Subset sum problem. Section 34.5.5, pages 1097-1100.
- Subgraph-isomorphism problem. Problem 34.5-1, page 1100.
- Set-partitioning problem. Problem 34.5-5, page 1101.
- Longest-simple-cycle problem. Problem 34.5-7, page 1101.
- Independent-set problem. Problem 34-1, pages 1101-1102.
- Graph-coloring problem. Problem 34-3, pages 1103-1104.
- Scheduling with profits and deadlines. Problem 34-4, pages 1104.

[CLRS] Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest, and C. Stein. MIT Press, 2009.

3. More Problems

- Exon chaining. Problem 6.29, pages 185-186 [DPV06].
- Reconstruction evolutionary trees by maximum parsimony. Problem 6.30, pages 186-187 [DPV06].
- Sequencing by hybridization. Problem 8.21, pages 268-269 [DPV06].
- Minimum Steiner Tree. Problem 9.6, page 294 [DPV06].
- Maximum Cut. Problem 9.9, page 295 [DPV06].
- Abductive inference (Diagnosis). Section 6.3, pages 245-254 [NN96].
- 0/1 knapsack problem. [DPV06].

[DPV06] Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. McGraw-Hill, 2006.
All the chapters of textbook can be found at:

http://www.cs.berkeley.edu/~vazirani/algorithms/all.pdf

[NN96] Foundations of algorithms by R. Neapolitan and K. Naimipour. D. C. Heath and Company, 1996.

4. Simulated Annealing

The main purpose of the projects in this category is to design, implement, and run a simulated annealing algorithm to solve an NP-hard problem. You can choose any problem from "2. NP-Hard problems from CLRS" or from "3. More Problems", or any NP-hard problem not previously mentioned. Once done, you should compare your results to the results obtained from running a publically available package.

5. Artificial Neural Networks

The main purpose of the projects in this category is to design, implement, and run an artificial neural network algorithm to solve an NP-hard problem. You can choose any problem from "2. NP-Hard problems from CLRS" or from "3. More Problems", or any NP-hard problem not previously mentioned. Once done, you should compare your results to the results obtained from running a publically available package.

6. Approximation Algorithm

The main purpose of the projects in this category is to design, implement, and run an approximation algorithm to solve an NP-hard problem. You can choose any problem from "2. NP-Hard problems from CLRS" or from "3. More Problems", or any NP-hard problem not previously mentioned. Once done, you should compare your results to the results obtained from running a publically available package.

7. Evolutionary Algorithms

The main purpose of the projects in this category is to design, implement, and run an evolutionary algorithm (genetic algorithm, genetic programming, and evolution strategy) to solve an NP-hard problem. You can choose any problem from "2. NP-Hard problems from CLRS" or from "3. More Problems", or any NP-hard problem not previously mentioned. Once done, you should compare your results to the results obtained from running a publically available package. The following is an additional problem where evolutionary algorithms have been successfully applied.

A) Craniofacial Superimposition in Forensic Identification

"Craniofacial superimposition is a process that aims to identify a person by overlaying a photograph and a model of the skull." [Ballerini, 2007] "Photographic supra-projection is a forensic process where photographs or video shots of a missing person are compared with the skull that is found. By projecting both photographs on top of each other (or, even better, matching a scanned three-dimensional skull model against the face photo/video shot), the forensic anthropologist can try to establish whether that is the same person." [Ibáñez, 2009].

This project involves reading the literature (use the 3 articles mentioned below as a starting point) and writing your own evolutionary algorithm-based program, modeled after their work, running and comparing your results to theirs.

[Ballerini, 2007] Ballerini, L., Cordón, O., Damas, S., Santamaría, J., Alemán, I., and Botella, M. Craniofacial superimposition in forensic identification using genetic algorithms, in: Proceedings of the IEEE International Workshop on Computational Forensics, Manchester, UK, 2007, pp. 429–434.

[Ibáñez, 2009] Ibáñez, O., Ballerini, L., Cordón, O., Damas, S., Santamaría, J. An experimental study on the applicability of evolutionary algorithms to craniofacial superimposition in forensic identification. Information Sciences 179 (2009) 3998–4028.

Nickerson, B., Fitzhorn, P., Koch, S., and Charney, M. (1991). A methodology for near-optimal computational superimposition of two dimensional digital facial photographs and three-dimensional cranial surface meshes. Journal of Forensic Sciences 36(2), 480–500.

8. Hidden Markov Models

The main purpose of the projects in this category is to design, implement, and run a hidden Markov Model algorithm to solve probabilistic problems such as the Dishonest Casino Problem. Once done, you should compare your results to the results obtained from running a publically available package.

9. Rough Sets

The main purpose of the projects in this category is to design, implement, and run a rough set algorithm to solve the 7-Segment Light Emission Diode Display or an NP-hard decision problem (or any NP-hard problem). You can choose any problem from "2. NP-Hard problems from CLRS" or from "3. More Problems", or any NP-hard problem not previously mentioned. Once done, you should compare your results to the results obtained from running a publically available package. The following is a definition of Rough Sets by the founder of the field, Zdzisław Pawlak.

"Rough set theory can be regarded as a new mathematical tool for imperfect data analysis. The theory has found applications in many domains, such as decision support, engineering, environment, banking, medicine and others. Rough set philosophy is founded on the assumption that with every object of the universe of discourse some information (data, knowledge) is associated. Objects characterized by the same information are indiscernible (similar) in view of the available information about them. The indiscernibility relation generated in this way is the mathematical basis of rough set theory. Any set of all indiscernible (similar) objects is called an elementary set, and forms a basic granule (atom) of knowledge about the universe. Any union of some elementary sets is referred to as a crisp (precise) set – otherwise the set is rough (imprecise, vague). Each rough set has boundary-line cases, i.e., objects which cannot be with certainty classified, by employing the available knowledge, as members of the set or its complement. Obviously rough sets, in contrast to precise sets, cannot be characterized in terms of information about their elements. With any rough set a pair of precise sets, called the lower and the upper approximation of the rough set, is associated. The lower approximation consists of all objects which surely belong to the set and the upper approximation contains all objects which possibly belong to the set. The difference between the upper and the lower approximation constitutes the boundary region of the rough set. Approximations are fundamental concepts of rough set theory." [Pawlak, 2003]

[Pawlak, 2003] "Rough set theory and its applications" by Zdzisław Pawlak; Journal of Telecommunications and Information Technology, 2003.

10. Implementing Games

- HEX

Hex is a board game played on a hexagonal grid, theoretically of any size and several possible shapes, but traditionally as an 11x11 rhombus. Other popular dimensions are 13x13 and 19x19 as a result of the game's relationship to the older game of Go.

Each player has an allocated color, Red and Blue being conventional. Players take turns placing a stone of their color on a single cell within the overall playing board. The goal is to form a connected path of your stones linking the opposing sides of the board marked by your colors, before your opponent connects his sides in a similar fashion. The first player to complete his connection wins the game. The four corner hexagons each belong to two sides.

To make it more challenging, you can also implement the "pie rule". Since the first player to move in Hex has a distinct advantage, the pie rule is generally implemented for fairness. This rule allows the second player to choose whether to switch positions with the first player after the first player makes the first move.

From: http://en.wikipedia.org/wiki/Hex_(board_game)

- Checkers

Checkers is played on the dark squares only. A piece may move one square at a time, diagonally. If one of your pieces is next to one of your opponent's pieces and the square beyond it is free, you are required to jump over the opponent's piece. The opponent's piece is then removed from the board. It is possible to jump many times in a row with the same piece, capturing several of your opponent's pieces. In the beginning, pieces can only move and jump forward. However, if a piece reaches the far end of the board (in the case of the person playing red, the top), then it becomes a king. (In checkers, a king is usually signified by stacking two checkers one on top of the other. In this program, the king has a star on it.) A king is allowed to move and jump diagonally backwards and forwards. Kings can be captured like any other piece. You win by capturing all of your opponent's pieces, or by blocking them so that they cannot move.

From: http://www.darkfish.com/checkers/Checkers.html

- The Game of LIFE

The life cellular automaton is run by placing a number of filled cells on a two-dimensional grid. Each generation then switches cells on or off depending on the state of the cells that surround it. The rules are defined as follows. All eight of the cells surrounding the current one are checked

to see if they are on or not. Any cells that are on are counted, and this count is then used to determine what will happen to the current cell.

1. Death: if the count is less than 2 or greater than 3, the current cell is switched off.

2. Survival: if (a) the count is exactly 2, or (b) the count is exactly 3 and the current cell is on, the current cell is left unchanged.

3. Birth: if the current cell is off and the count is exactly 3, the current cell is switched on.

From: http://mathworld.wolfram.com/GameofLife.html

- Other games, such as Chess, Othello (Reversi), GO, Jenga, Tetris and Chess.

11. Data Compression Algorithms

11.1. Dictionary-Based Data Compression Techniques

There are several approaches to data compression. This project concentrates on the dictionary-based approach.

The original dictionary-based data compression methods were the two Lempel-Ziv methods known today as LZ77 and LZ78. The latter was improved by Welch, resulting in the popular LZW method. Quite a few variants of these methods are known today, the most popular being LZSS, LZMW, LZAP, LZY and LZP.

The project involves literature search, studying two or three of the above variants, and implementing one of them.

References:

D. Salomon, G. Motta, and D. Bryant, Data Compression: The Complete Reference, Fourth Edition, 2007.

T. Bell, J. Cleary, and I. Witten, Text Compression, Prentice Hall, 1990.

Mark Nelson et al., The Data Compression Book, M&T Books, 2nd edition, 1996.

Sayood, K. Introduction to Data Compression, Third Edition, Morgan Kaufmann Publishers, 2005.

11.2 PC and UNIX Data Compression Programs

There are many data compression methods for all common computers but they are based on just a few principles.

The three main principles used for compressing data are run-length encoding, Huffman codes, and LZ methods (dictionary-based). Almost all

common data compression programs such as ARC, Zip, PKzip, LHarc and ARJ for the PC, and compress for UNIX, use variants or combinations of those principles.

The project consists of two parts. The first, larger part involves a literature search to find how the popular data compression programs work, how they are based on the basic principles listed above, and what techniques and data structures they use. The second part is an implementation of one of these. It should be simple and readable, and does not have to be fast, efficient or fancy.

See Bibliography for 11.1.

11.3 Scrambling as a Prefix to Data Compression

A useful approach to data compression is to scramble the data before it is compressed. The scrambling should satisfy two conditions:

1. It should be reversible (otherwise it is impossible to decode the compressed file).

2. It should transform the original data to a more correlated form. In the case of text compression, the scrambled data should have concentrations of identical characters. In the case of image compression, scrambling should bring pixels of the same color close together.

One approach to scrambling is the Burrows-Wheeler transform. It transforms any data—text, images and sound—to a more correlated form that can later be highly compressed using traditional methods. Another approach—for image compression—is to scan the image along a space-filling curve instead of by scan lines.

This project involves both literature search and implementation and testing of one algorithm (such as Burrows-Wheeler).

See Bibliography for 11.1

11.4 Context-Based Text Compression.

Statistical methods for text compression assign variable-size codes to the text characters, such that common characters are assigned short codes. The problem is that every document has a different distribution of characters. The letter 'E', e.g., occurs most often in English texts, but each document may have a different percentage of 'E's in it.

Modern text-compression methods are thus adaptive. They adjust themselves to the frequencies of characters in the particular document being compressed 'on the fly.' If the letter 'E', e.g., does not occur very often at the beginning of a particular document, it may be initially assigned a long code. If it appears more often later in the document, it may be reassigned a shorter code.

A context-based adaptive compression method goes one step further. It takes into account not just the frequency of occurrence of a character, but the probability that the character will appear in a certain context. If the last characters to be read and compressed were, e.g., "rea", then the algorithm has to find how many times this string was seen in the past, and how many times it was followed by 'l', by 'p' and so on. Based on this data, the algorithm decides what probability to assign to the next character, and how to encode it.

This is a complex method that involves implementing arithmetic coding as part of the overall project. However, it is ideal for someone who is seriously interested in data compression since it gives an insight on how modern compression methods work.

This project involves both literature search and implementation and testing of one adaptive algorithm, such as the Adaptive Huffman Algorithm.

See Bibliography for 11.1.

12. Fast Fourier and Other Similar Transforms

- Hadamard Transforms and applications.
- Walsh Transforms, including fast Walsh transforms and discrete Walsh transforms, and their applications.
- Haar Transforms, including fast Haar transforms and discrete Haar transforms, and their applications.

13. Visualization/Animation Algorithms

The main purpose of the projects in this category is to develop a visualization or animation tool to assist students in learning how the algorithm works. One should be able to use the interactive, user-friendly, educational tool you are to develop, in classroom demonstrations, hands-on laboratories, self-directed work outside of class, and distance learning. The visualization package will include background material, a detailed explanation of the algorithm, with examples, and quizzes and exercises. Using Java is highly recommended.

For these projects you have to implement visualization or an animation for one of the algorithms that we discussed in class, or an algorithm related to the topics covered in this course.

13.1. Image Compression by DCT or Wavelet Transforms

JPEG is a lossy compression algorithm that has been conceived to reduce the file size of natural, photographic-like true-color images as much as possible without affecting the quality of the image as

experienced by the human sensory engine. We perceive small changes in brightness more readily than we do small changes in color. It is this aspect of our perception that JPEG compression exploits in an effort to reduce the file size. [WWW1]

Image files tend to be large, so users are constantly looking for efficient image compression methods. Images can be greatly compressed because:

(1) a typical image tends to have much redundancy, and

(2) some image information can be lost in the compression, without the user noticing any degradation of image quality when the image is decompressed and displayed.

The discrete wavelet transform is used in several modern image compression methods and achieves excellent results, especially for lossy compression. The idea is to identify the various frequencies in different parts of the image, and to delete the highest frequencies in each part.

Mastering wavelet transforms is beyond the scope of this project, but it is possible to understand the principles and to implement some useful code by concentrating on the simplest wavelet transforms.

The project is a JAVA-based visualization package that illustrates the principles of wavelet image compression through the use of the

- Discrete Cosine Transform, or
- Haar Wavelet Transform.

Simple code should be written to demonstrate how this method results in subbands that reflect the various frequencies of pixels in the image.

Reference: www.cs.sjsu.edu/faculty/khuri/publications.html

S. Khuri, H. Hsu, Interactive Packages for Learning Image Compression Algorithms, Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education, July, 2000, pp. 73-76.

[WWW1] www.prepressure.com/techno/compressionjpeg.htm

13.2. Elementary Graph Algorithms

Design and implement a visual interactive software package to demonstrate how the different graph algorithms described in Chapter 22 of [CLRS].

[CLRS] Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest, and C. Stein. MIT Press, 2009.

14. Probabilistic Graphical Models

The main purpose of the projects in this category is to design, implement, and run a probabilistic graphical model algorithm to solve an NP-hard decision problem (or any NP-hard problem).

According to K Murphy (www.cs.ubc.ca/~murphyk/Bayes/bnintro.html): Probabilistic graphical models (PGMs) are graphs in which nodes represent random variables, and the (lack of) arcs represent conditional independence assumptions. Hence they provide a compact representation of joint probability distributions. Undirected graphical models, also called Markov Random Fields (MRF) or Markov networks, have a simple definition of independence: two (sets of) nodes A and B are conditionally independent given a third set, C, if all paths between the nodes in A and B are separated by a node in C. By contrast, directed graphical models, also called Bayesian Networks or Belief Networks (BN), have a more complicated notion of independence, which takes into account the directionality of the arcs. Undirected graphical models are more popular with the physics and vision communities, and directed models are more popular with the AI and statistics communities. A graphical model specifies a complete joint probability distribution (JPD) over all the variables. Given the JPD, we can answer all possible inference queries by marginalization (summing out over irrelevant variables). However, summing over the JPD takes exponential time. PGMs are full of many efficient algorithms.

This project involves reading the literature (use the first three references: [1], [2], and [3], mentioned below as a starting point) and writing your own PGM-based program, that solves one of many possible applications, such as, medicine, meteorology, speech recognition, intelligent tutoring, gambling, monitoring, games (Sudoku [4], [5]), constraint satisfaction problems [6], low-density parity-check codes [7], and text classification [8]. Then, you should run your program and compare your results with those obtained from running publically available packages.

[1] Probabilistic Graphical Models, Coursera & Stanford – Daphne Koller, last retrieved on January 1, 2014.

https://class.coursera.org/pgm/lecture/preview

[2] Koller D. & Nir Friedman. Probabilistic Graphical Models. MIT press ISBN 978-0262013192

[3] Kevin Murphy; www.cs.ubc.ca/~murphyk/Bayes/bnintro.html

[4] Khan, Sheehan; Jabbari, Shahab; Jabbari, Shahin; Ghanbarinejad, Majid., “Solving Sudoku Using Probabilistic Graphical Models”.

[5] Goldberger, J., "Solving Sudoku Using Combined Message Passing Algorithms", Technical Report TRBIU-ENG-2007-05-03, Engineering School, Bar-Ilan Univ. (2007).

[6] A. Braunstein, M. Mezard and R. Zecchina. Survey propagation: an algorithm for satisfiability. Random Structures and Algorithms 27, 201-226. (2005).

[7] C. Di, D. Proietti, T. Richardson, E. Telatar, and R. Urbanke. Finite length analysis of low-density parity-check codes on the binary erasure channel. IEEE Trans. on Info. Theory, pp. 1570-1579. (2002).

[8] http://www.stanford.edu/class/cs124/lec/naivebayes.pdf.

15. Fragment Assembly Problem

To sequence a DNA molecule is to obtain the string bases (A, C, G, or T) that it contains. In large scale DNA sequencing, we have to sequence large DNA molecules (hundreds of thousands of base pairs). It is impossible to directly sequence contiguous stretches of more than a few hundred bases. On the other hand, we know how to produce enough copies of the molecule to sequence and how to cut random pieces of a long DNA molecule. The problem is that these pieces (fragments) have to be assembled. Thus, the fragment assembly problem consists in reconstructing a DNA sequence (a sequence of characters over the alphabet {A,C,G,T}) from a collection of randomly sampled fragments.

This project consists in:

1. Choosing an algorithm, such as the greedy algorithm (Section 4.3.4 in [SM97]), or any other heuristic, such as the ones described in Section 4.4 of [SM97] and in [KS99], and in Sections 8.3 to 8.9, pages 262 to 280 in [JP04],

2. Reading, understanding and implementing the algorithm you choose,

3. Comparing your program's performance to an existing implementation, e.g. phrap, cap, etc.

[JP04] Jones, N., and Pevzner, P. Bioinformatics Algorithms. MIT Press, 2004.

[KS99] Kim, S., and Segre, A., AMASS: A Structured Pattern Matching Approach to Shotgun Sequence Assembly. Journal of Computational Biology, 6(2), 1999, pp 163-186; 1999.

[SM97] Setubal, J. and Meidanis J. Introduction to Computational Molecular Biology. PWS Publishing Company, 1997.

16. Genome Rearrangement

Various global rearrangements of permutations, such as reversals and transpositions, have recently become of interest because of their applications in computational molecular biology. A reversal is an operation that reverses the order of a substring of a permutation. The problem of determining the smallest number of reversals required to transform a given permutation into the identity permutation is called sorting by reversals. This problem is of interest because the permutations can be used to represent sequences of genes in chromosomes, and the global rearrangements then represent evolutionary events. As a result, these problems are called genome rearrangement problems. Genome rearrangement problems seem to be unlike previously studied algorithmic problems on sequences, so new methods have had to be developed to deal with them. [DC98]

This project consists in choosing an algorithm, such as the ones described in Section 7.2 of [SM97], in Chapter 5 in [JP04], and in [DC98]; reading, understanding and implementing the algorithm you choose and testing it on several different input data.

[DC98] Genome Rearrangement Problems. Ph.D. Dissertation by David Alan Christie. University of Glasgow, Scotland. 1998.

[JP04] Jones, N., and Pevzner, P. Bioinformatics Algorithms. MIT Press, 2004.

[SM97] Setubal, J. and Meidanis J. Introduction to Computational Molecular Biology. PWS Publishing Company, 1997.

