Programming Projects (updated: Jan 21, 2007)

Every student is expected to do a project that uses or develops techniques, theory, or algorithms introduced in the class. In previous years, a few class projects have resulted in MS theses and even in conference publications. 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. The Internet can be a big help for tracking down ideas.

You can

- pick a project from the list,
- modify a project to suit your interests, or
- propose 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, 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.

A two-page project proposal is due by the beginning of the lecture on Tuesday, March 13, 2007. The proposal should give a clear description of the project and should contain absolutely no generalities and definitions. Clearly state what you are planning to do and explain how you plan to achieve it. Do not forget to specify the programming language you intend to use. It is important to get started as early as possible.

The projects are due at on Tuesday, May 8, 2007. Make sure to hand in both a technical project description with appendices and references, and a disk or a CD containing the source code. The approximate length of the programming project report should be 10 pages. You do not need to include a hard copy of the source code.

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.
List of Suggested Projects

1. **Sudoku**

The name "Sudoku" is the Japanese abbreviation of a longer phrase, "suji wa dokushin ni kagiru", meaning "the digits must occur only once". Sudoku is a logic-based placement puzzle. The objective is to fill a 9x9 grid so that each column, each row, and each of the nine 3x3 boxes contains the digits 1 to 9. The puzzle setter provides a partially completed grid, so that there is only one solution [WWW1].

There are many different ways of solving the problem. In this project, you will choose an algorithm, implement it and test it.


2. **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],
2. reading, understanding and implementing the algorithm you choose,
3. comparing your program’s performance to an existing implementation, e.g. phrap, cap, etc.


3. Problems from our textbook
Pick any problem and choose an algorithm to implement and test. A few suggestions:
• Assembly-line scheduling. Section 15.1, pages 324-331.
• Matrix-chain multiplication. Section 15.2, pages 331-339.
• Longest common subsequence. Section 15.4, pages 350-356.
• A task-scheduling problem. Section 16.5, pages 399-401.
• Scheduling to minimize average completion time. Problem 16-2, pages 402-403.

4. NP-Hard problems from our textbook
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, page 1003.
• Vertex cover problem. Section 34.5.2, page 1006.
• Hamiltonian cycle problem. Section 34.53, page 1008.
• Traveling salesperson problem. Section 34.5.4, page 1012.
• Subset sum problem. Section 34.5.5, pages 1013-1014.
• Subgraph-isomorphism problem. Problem 34.5-1, page 1017.
• Set-partitioning problem. Problem 34.5-5, page 1017.
• Longest-simple-cycle problem. Problem 34.5-7, page 1017.
• Graph-coloring problem. Problem 34-3, pages 1019-1020.
• Scheduling with profits and deadlines. Problem 34-4, pages 1020-1021.
5. 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. Lecture Notes.


A draft of the entire textbook can be found at:
http://www.cse.ucsd.edu/users/dasgupta/mcgrawhill/all.pdf


6. 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.

6.1. 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

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

**6.2. Elementary Graph Algorithms**

Design and implement a visual interactive software package to demonstrate how the different graph algorithms described in Chapter 22 of our textbook work.