Pursuit and Evasion

Bachelor Thesis

by

Felix Blumenberg
870605-0633
Fredrik Båberg
880312-0511
Mats Malmberg
860802-0338

under the guidance of

Doctoral Student Johan Thunberg
and
Professor, Ph.D. Xiaoming Hu

Department of mathematics
Division of Optimization & Systems Theory
Royal Institute of Technology KTH, Sweden
Feb 2011

Abstract

Abstract in ENGLISH

Sammanfattning

Abstract på SVENSKA

Contents

List of Figures

Chapter 1

Introduction

1.1 Objective

The goal of this research is to solve a pursuit evasion problems in a two dimensional environment, with the help of heuristic problem solving methods. In short, the problem is: given a surface with static obstacles and a number of pursuers ensure that an evader cannot exist within the area. A complete formulation of the problem can be seen in Chapter 2

The problem itself is solved. An algorithm providing an optimal solution already exists. The problem yet to be solved, is the problem of finding a solution within a reasonable computational time, it is here where heuristic methods comes in to play

With great risk for loss of optimality, our goal is to use heuristic methods to find a solution with a reasonable computational time

This report is therefore a study of the balance between a reasonable computational time and the loss of optimality.

1.2 Organization Of the Report

1. Chapter 2 contains a detailed formulation of the problem and are approach to it.
2. Chapter 3 explains the simulation environment we built.
3. Chapter 4 contains a description of the algorithms we decided to use.
4. Chapter 5 contains the results of our simulations.
5. Chapter 6 contains a discussion of our results, comparison and conclusions.

1.3 Related work

Relaterat arbete bla de tre givna articklarna

1.4 Background

1.4.1 What is optimization

Optimization is a mathematical discipline, for solving various types of problems. With the aim of finding the best, available, values of some objective function given a defined domain. If one wants to know more advise NAME ON LITETUR

optimization theory makes use of various so-called, mathematical programming models, to set up and solve practical problems. Linear optimization problems are treated by means of Linear Programming LP, non-linear optimization problems using Non-linear Programming NLP. And integer optimization problems with Integer Programming IP. A further breakdown of the type of problems one deals with may be done by studying its complexity.

complexity is more or less computationl time. There are many different complexity classes of problems, but two of the most fundamental is the P and NP

1.4.2 P & NP problems

P is the set of problems which can be solved by a deterministic Turing machine using a polynomial amount of computation time. Cobham?Edmonds thesis holds that P is the class of computational problems which are "efficiently solvable" or "tractable". In practice, some problems not known to be in P have practical solutions, and some that are in P do not, but this is a useful rule of thumb.

NP refere to Nondeterministic Polynomial time. and can be intuitively described as the set of problems for which, a found solution can be verified to actually be the correct solution, using a deterministic Turing machine using a polynomial amount of computation time.

A set of descriptions that leads one to wonder whether P = NP In what the essence is: Suppose that solutions to a problem can be verified quickly. Then, can the solutions themselves also be computed quickly? A unsolved question considered by many to be the most important problem in the feeld in computer science.

Within the class NP there are a few subclasses where the hardest problems among the subclasses are called NP-complete. With are problems of the type for which no polynomial-time algorithms are known. As stated above there is still an open question whether such algorithms actually exist for NP-complete problems, and by corollary, all NP problems. It is widely believed that this is not the case [referens 1]

This is very relevant to our problem, related work has shown that the problem in question is at least NP-hard [reference 2] wich is a set of problems said to be a class of problems That, informally are, at least as hard as the hardest Problems in NP.

Preliminara ej fixade referenser REFERENS

[1] William I. Gasarch (June 2002). "The P=?NP poll." (PDF). SIGACT News 33 (2): 34?47. doi:10.1145/1052796.1052804. http://www.cs.umd.edu/ gasarch/papers/poll.pdf. Retrieved 2008-12-29.

[2] Deras 3 articklar.

1.4.3 (the need for) a near optimal solution

Due to the fact that the problem in question is NP-hard and thus the possibilities of finding an optimal solution within reasonable computing time are/is seen to be very low. Does the logical thread has brought us to be prepared to sacrifice optimality This sacrifice leads us to a larger spectrum of possible approaches to the problem.

1.4.4 Heuristic methods

Heuristic methods are a bransch of methods used in computer science and mathematics to make educated guesses to help find solutions to a problem. Generally heuristic methods douse not provide correct / optimal solutions but it offers a reasonably fast computed solution. Heuristic methods are therefore often used, very powerful, in combination with deterministic methods, to speed up the process.

The characteristics of a heuristic is partly that there is no evidence of that it works, or else it would be a deterministic method. The idea of heuristic methods is thus to improve the chances of finding the desired solution. Pure random guesses forms no heuristic method.

There are several occasions where a heuristic method is preferable to a deterministic method. Here are a few:

- There is no known algorithm for solving a specific problem, a heuristic is the only way to approach a solution

- A good algorithm can be difficult to implement and heuristics can sometimes be accepted because it is easy to describe in a programming language and that heuristic methods are known to produce good results.

- The problem is too difficult to solve efficiently and quickly with deterministic methods. A heuristic method could overcome that and give an acceptable but probably not an optimal solution

Where the last point corresponds to the problem we are dealing with.
Consept misunderstanding?: Computationl time vs computation time

Chapter 2

Problem formulation

The essentials of our problem formulation today is "To test implementation of greedy approach, tabu search and genetic programming and evaluation." The exact formulation is in progress...

2.1 Problem with the problem

the headline is self explanatory, the ide her is to discuss the P vs NP aspekt. maybe this will be incorporated with another part of the introduction

2.2 Approach

A description of our approach to the problem. Describing the overview of our approach. Mentioning the creation of an testing enviroment, choice and application of heuristics and target data to evaluate the effectiveness.

Chapter 3

Simulation Environment

In order to attain the data needed for a comparison of our different algorithms it was nessecary to construct a good testing enviroment. We decided to create this enviroment by the use of two separate parts. One part is called the "Map generator". This part creates a map of the enviroment, tests the feasibility and prints feasible enviroments into an output file. The other part is called the "Network generator". This part reads in an enviroment from a file, creates a graph network to the corresponding map and gives each node in the network its relevant information.

3.1 Map generator

The Map Generator (MG) creates random feasible enviroments. A feasible enviroment is described more in detail in section 2, but in short one could say that an enviroment is feasible if it is simply connected and can be divided into a finite set of convex regions. Given the desired size and the density (percentage of obstacles per total area) as inputs, MG creates square shaped feasible enviroment with randomly placed obstacles and saves the map in an external file. For simplicity we have chosen to construct enviroments consisting only of square regions. We suggest that this does not result in a loss of generality since one can construct any feasible enviroment by a sufficiently fine meshing of squares.

pseudo-code, Map Generator:
input variables:
int size; //Specifies the width and height of the square matrix A.
int NumberOfEnv; // Specifies how many feasible enviroments to create.
int Obstacle; // Specifies the number of obstacles in percents, e.g. number of obstacles per total area of A.

while (created feasible enviroments is less than the variable NumberOfEnv)

Create a matrix A with all elements set to one;
in a random maner, turn elements in A to zero (corresponding to obstacles) until the desired amount of obstacles have been placed.
Test if A is connected
if A is connected write the matrix A to the file OK.out
if A is not connected write the matrix to a file NotOK.out

3.2 Network Generator

The Network Generator (NG) generates a node network from an environment matrix. Each node contains data of its adjacent nodes, all the nodes visible from it and its current state. For input the NG uses an environment matrix that is either created by the MG or by hand.

Pseudo-code, Node Network Generator:
    Read environment matrix A
    Represent each element in A with a node in a matrix B
    For each node in B:
        Set name to its position in the matrix
        Find and store all nodes that is adjacent to the current node
        Find and store all visible nodes

Chapter 4

Methods

A Description of the methods and algorithms we have used.

4.1 Greedy

Greedy

4.1.1 Description

A description of what Greedy is

4.1.2 Algorithm

A description of the algorithm

4.1.3 Implementation

A description of how the algorithm is implemented

4.1.4 Development process

How the development of the algorithm have proceeded.

4.2 Tabu

Tabu

4.2.1 Description

A description of what Tabu is

4.2.2 Algorithm

A description of the algorithm

4.2.3 Implementation

A description of how the algorithm is implemented

4.2.4 Development process

How the development of the algorithm have proceeded.

4.3 Genetic

4.3.1 Description

Genetic algorithms is based on the idea of evolution. Using a combination of reproduction, mutation and survival of the fittest a solution is generated.

4.3.2 Algorithm

A description of the algorithm

4.3.3 Implementation

A description of how the algorithm is implemented

4.3.4 Development process

How the development of the algorithm have proceeded.

Chapter 5

Results

Statistics, tables and a description of the tables. Also why we have chosen these tables etc.

Results for MILP evaluation could also be added here.

Chapter 6

Discusion

This chapter contains analysis of each algorithm, why it did or did not work, how it compares to the other algorithms and a conclusion. The main purpose is to present conclusions from the data presented in the previous chapter.

6.1 Analysis

An analysis of each algorithm, evaluation of why it did or did not work.

6.2 Comparsion and statistics

A comparsion between the algorithms, and perhaps also with MILP.

Chapter 7

Conclusion

A conclusion of our work, and future work.

Bibliography

[1] Abraham Silberschatz, Peter Baer Galvin, "Operating System Concept", Addison Wesley, Reading Massachusetts, USA, 1998

[2] John P. Hayes, "Computer Architecture and Organization", McGraw-Hill International Company, Singapore, 1988

[3] PVM 3 User Guide and Reference Manual, Edited by Al Gist, Oak Ridge National Laboratory, Engineering Physics and Mathematics Divison, Mathematical Science Section, Oak Ridge, Tennessee, USA, 1991

[4] PVM's HTTP Site, "http://www.epm.ornl.gov/pvm/"

[5] Brian W. Kernighan, Dennis M. Ritchie, "The C - Programming Language, (ANSI C Version)", Prentice-Hall of India Pvt. Ltd., New Delhi, 1998

[6] Thomas H. Corman, Charles E. Leiserson, Ronald L. Rivest, "Introduction to Algorithm", MIT Press, Cambridge, MA, USA, 1990

[7] Kenneth Hoffmann, Rey Kunze, " Linear Algebra", Prentice-Hall of India Pvt. Ltd., New Delhi, 1997

[8] G.H. Golub and C. F. Van Loan , " Matrix Computations", Third Edition. The Johns Hopkins University Press, Baltimore, 1996

[9] David A. Patterson, John L. Hennessy, "Computer Architecture, A Quantitative Approach", Morgan Kaufmann Publications Inc., San Mateo, California, USA, 1990

[10] Jack Dongarra, Iain Duff, Danny Sorensen, and Henk van der Vorst, Numerical Linear Algebra for High-Performance Computing",Society for Industrial and Applied Mathematics, Philadelphia, 1998