Solving Binary Optimisation Problems using a Binary Genetic Algorithm

Shan He

School for Computational Science
University of Birmingham

Module 06-27818 and 27819: Advanced Aspects of Nature-Inspired Search and Optimisation (Ext)

Outline of Topics

Capital budgeting/project selection problem

- A company is evaluating 4 projects which each run for 3 years and have the following characteristics.

		Capital requirements (m)			
Project	Return (£m)	Year	1	2	3
1	0.2		0.5	0.3	0.2
2	0.3		1.0	0.8	0.2
3	0.5		1.5	1.5	0.3
4	0.1		0.1	0.4	0.1
Available capital (£m)			3.1	2.5	0.4

- Decision problem: Which projects should be selected to maximize the total profits?

Capital budgeting/project selection problem

Explanation:

- Once a project has been selected, all yearly capital requirement (investments) must be met in full
- There is a capital (budget) available for each year, that should not be exceeded

Problem formulation

- **Decision variables** $\mathbf{x} = [x_1, x_2, x_3, x_4]$: which project to select
  - $\mathbf{x} \in \{0, 1\}$
  - $x_i = 0$: project i is not selected
  - $x_i = 1$: project i is selected
  - It is essentially a **0-1 (binary) integer programming problem**
- **Objective**: Maximize the expected returns

$$\texttt{maximise}\ 0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$$

- **Constraints**: Yearly budget cannot be exceeded

$$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 <= 3.1$$
$$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 <= 2.5$$
$$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 <= 0.4$$

0-1 integer programming problems

0-1 integer programming is NP-complete

Imaging we have a simple decision making problem with a yes or no answer.

- P vs NP vs NP-complete vs NP-hard
  - **P**: a complexity class that represents the set of all decision problems that can be **solved** in polynomial time (efficiently).
  - **NP**: a complexity class that represents the set of all decision problems for which the instances where the answer is "yes" have proofs that can be **verified** in polynomial time.
  - **NP-complete**: NP-Complete is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time.
    - Intuition: NP-complete means we can solve Y quickly if we know how to solve X quickly.

0-1 integer programming problems

0-1 integer programming is NP-complete

- P vs NP vs NP-complete vs NP-hard
  - **P versus NP problem**: an 1-million dollar open question to ask whether "polynomial time algorithms actually exist for solving NP-Complete, or NP problems? Current answer is NO.
    - "*NP-complete this is widely regarded as a sign that a polynomial algorithm for this problem is* **unlikely** *to exist*"
    - What if we can proof P = NP: "*What we would gain from P = NP will make the whole Internet look like a footnote in history.*"
  - Richard Karp's 21 NP-complete problems
  - **NP-Hard**: the problems that are at least as hard as the NP-complete problems.
    - Note: NP-hard problems do not have to be in NP, and they do not have to be decision problems.

How to solve the problem

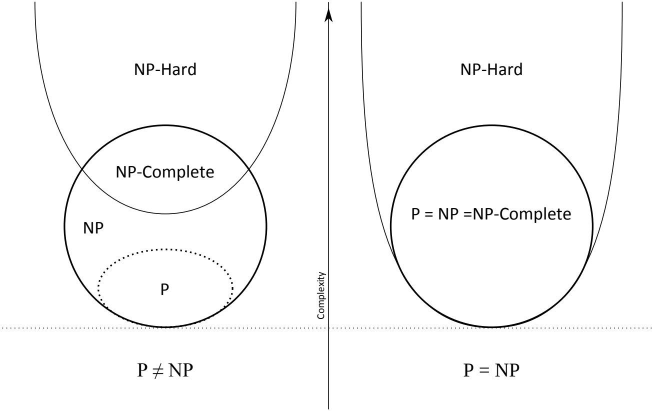

Figure: Euler diagram for P, NP, NP-complete, and NP-hard set of problems. From Wikipedia

How to solve the problem

- Deterministic (Exact) methods:
    - The Branch & Bound Method: Relax the problem as a continuous problem
        - The Branch & Bound Method: Relax the problem as a continuous problem
- Heuristic methods:
    - Local search (hill climbing)
    - Simulated annealing
    - **Genetic Algorithm**

Generic Genetic Algorithm

Generic Genetic Algorithm

$\mathbf{X}_0$:= generate initial population of solutions
terminationflag := false
t := 0
Evaluate the fitness of each individual in $\mathbf{X}_0$.
while (terminationflag != true)
    Select parents from $\mathbf{X}_t$ based on their fitness.
    Breed new individuals by applying crossover and mutation to parents
    Evaluate the fitness of new individuals.
    Generate population $\mathbf{X}_{t+1}$ by replacing least-fit individuals
    t := t + 1
    If a termination criterion is met: terminationflag := true
Output x_{best}

Building blocks of Evolutionary Algorithms

- An Evolutionary Algorithms consists of:
  - **representation**: each solution is called an individual
  - **fitness** (objective) function: to evaluate solutions
  - **variation operators**: mutation and crossover
  - **selection and reproduction** : survival of the fittest

Genetic Representation: general principle

- The selection of representation depends on the problem
- We have the following choices:
  - **Binary representation**
  - **Real number representation**
  - **Random key representation**
  - Other problem specific representations

Exercise: BGA for project selection problem

- Open my BGA_template.m file
- Understand the fitness calculation function
- Complete the following sections:
  - Initialisation
  - Crossover
  - Mutation
- Run you algorithm 30 times and record the average results and standard deviation.

Exercise: Project selection problem extensions

- A new scenario:
  - If the company select project 4, the return will be realised and can be reinvested in year 3.
  - Projects 1 and 2 are mutually exclusive
- How to model this problem?

Exercise: Project selection problem extensions

- Solution:
  - If the company select project 4, the return will be realised and can be **reinvest in year 3**.

$$\texttt{maximise}\ 0.2x_1 + 0.3x_2 + 0.5x_3$$

and for year 3, the constraint becomes

$$0.2x_1 + 0.2x_2 + 0.3x_3 <= 0.4 + 0.1x_4$$

or

$$0.2x_1 + 0.2x_2 + 0.3x_3 - 0.1x_4 <= 0.4$$

  - Projects 1 and 2 are mutually exclusive:

$$x_1 + x2 <= 1$$

- Please read Prof. J E Beasley's OR-Notes on Integer Programming

Extra exercise

- Use your implemented Binary GA to solve the office assignment problem