Master of Technology

Computational Intelligence II

Modelling Problems for GA Solutions

Dr. Zhu Fangming
Institute of Systems Science,
National University of Singapore
Email: isszfm@nus.edu.sg

Objectives

- To illustrate how to model problems for GA solutions

Knapsack Problem

- A thief robbing a store finds n items: the *i*th item is worth P_i dollars and weighs W_i kg. He wants to take as valuable a load as possible, but he can carry at most *C* kg in his knapsack. What items should he take?

- This is 0-1 knapsack problem as each item must either be taken or left behind -- the thief cannot take a fractional amount of an item or take an item more than once.

Knapsack Problem

- GA Chromosome:

  Candidate Solutions can be represented as vectors:

  $S=(s_1, \ldots , s_n)$

  where s_i is 1 if taken, and 0 if not.

1	**0**	**0**	**1**	**…**	**0**	**1**

- Fitness Function:
  - Maximize the total value of the items taken
- Constraints:
  - Maximum *C* kg in knapsack

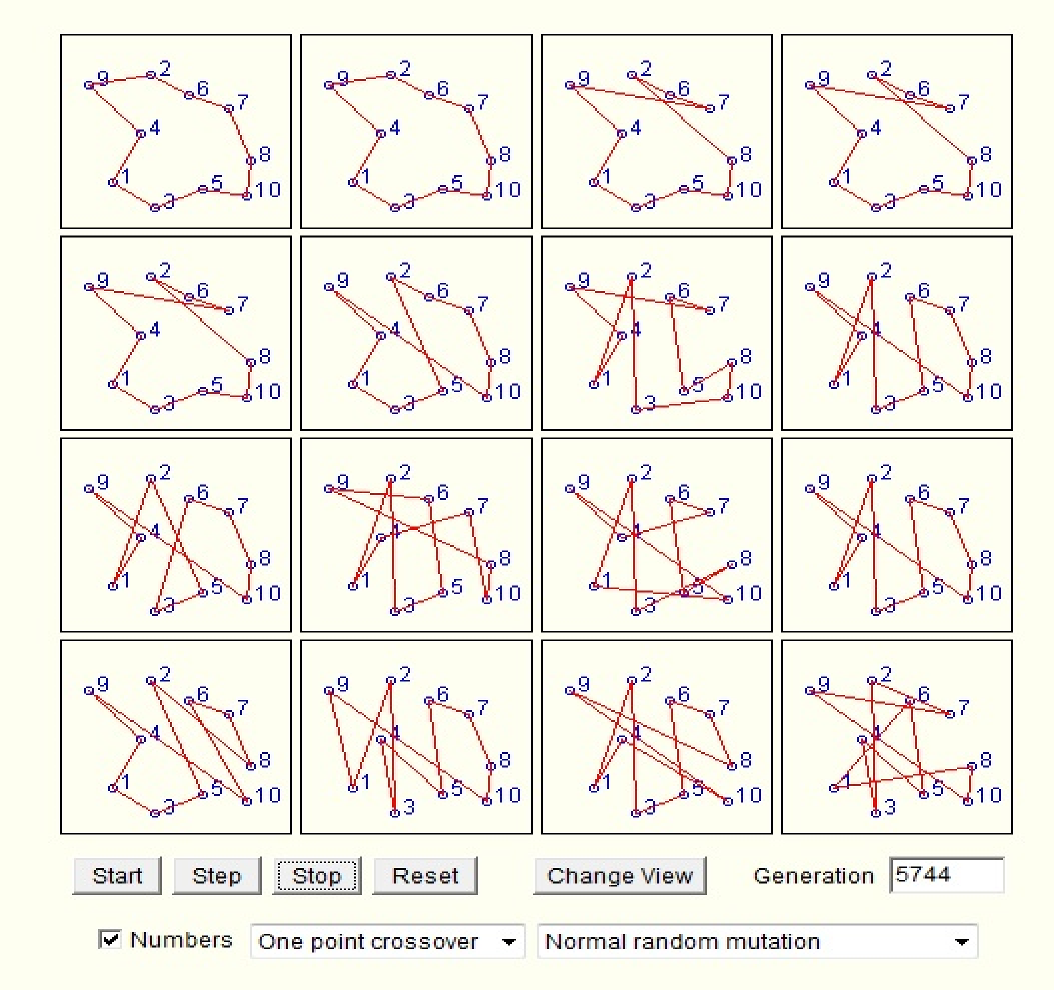
Traveling Salesman Problem

- The Traveling Salesman Problem (TSP) is a famous problem.
- The traveling salesman must visit every city in his territory exactly once and return to the starting point.
- Given the cost of travel between all cities, how should he plan his itinerary for minimum total cost of the entire tour?

The tour of 13,500 US cities

http://www.tsp.gatech.edu/

Traveling Salesman Problem



http://www.obitko.com/tutorials/genetic-algorithms/tsp-example.php

Traveling Salesman Problem

- GA Chromosome:

(J H C I G F B D E A) or (7 6 5 1 4 2 3 10 9 8)

- Fitness Function:
  - Minimize the total cost (or identify the shortest path)

- Constraints:
  - Must visit every city exactly once and return to the starting point

Airline Crew Scheduling

- Next to fuel cost, personnel cost is the largest cost an airline faces.
- The effective use of personnel can have a tremendous impact on the bottom line.
- There are, however, a tremendous number of restrictions on how airline personnel are used. Determining a feasible crew schedule is a very difficult problem.
- Years ago, American Airlines was interested in applying OR techniques to crew scheduling.

Example Formulation

- **Flights/Legs to be covered**:
  1. SIN 9-12 HK
  2. HK 13-15 BEIJING
  3. BEIJING 16-18 TOKYO
  4. BEIJING 17-19 SIN
  5. TOKYO 19-21 HK
  6. TOKYO 19-21 SIN
  7. HK 14-16 BEIJING
  8. SIN 16-18 TOKYO

Schedules/Pairings:

1. SIN 9-12 HK 14-16 BEIJING 17-19 SIN (1,7,4)
2. HK 13-15 BEIJING 16-18 TOKYO 19-21 HK (2,3,5)
3. BEIJING 16-18 TOKYO 19-21 SIN 9-12 HK 13-15 BEIJING (3,6,1,2)
4. SIN 16-18 TOKYO 19-21 SIN (8,6)
5. SIN 16-18 TOKYO 19-21 HK 14-16 BEIJING 17-19 SIN (8,5,7,4)

Airline Crew Scheduling

- A variable is created for each feasible schedule for a single crew. For instance, a schedule where the crew leaves SIN at 9:00, arrives in HK at 12:00, leaves HK at 14:00, arrives in BEIJING at 16:00, leaves BEIJING at 17:00 and arrives in SIN at 19:00 would correspond to a single variable, x1, say.
- The cost of for each feasible schedule is also known.
- The constraint is simple - There must have exactly one schedule to cover each flight/leg.

SPP (Set Partition Problem) model

Minimize $z = \sum_{j=1 \text{ to } n} c(j)*x(j)$

subject to $\sum_{j=1 \text{ to } n} a(i,j)*x(j) = 1$ for $i = 1, \ldots, m$

$x(j) = 0$ or 1 for $j = 1, \ldots, n$

Example SPP problem

	C1=35	C2=30	C3=50	C4=40	C5=60	
F1	1	0	1	0	0	**1**
F2	0	1	1	0	0	**1**
F3	0	1	1	0	0	**1**
F4	1	0	0	0	1	**1**
F5	0	1	0	0	1	**1**
F6	0	0	1	1	0	**1**
F7	1	0	0	0	1	**1**
F8	0	0	0	1	1	**1**
	X1	**X2**	**X3**	**X4**	**X5**	

Example SPP problem

- 8 flights & 5 schedules
- First line shows the cost coefficients of each schedule
- Last line shows the indices of the schedules
- A feasible solution: $x3 = x5 = 1$, and the other $xj = 0$ with $z = 110$
- An infeasible solution: $x1 = x5 = 1$, and the other $xj = 0$

Applications of SPP

- A wide variety of practical applications have been modeled as SPPs during the past 50 years, including
  - Airline and Bus Crew Scheduling,
  - Vehicle Routing,
  - Circuit Partitioning

Constraints

- SPP is a highly constrained problem
- Just finding a feasible solution to the SPP is very hard
- It is likely, in the initial stages, many or most strings in the population are infeasible

Two Choices of Penalty Terms

Minimize $p(x) = \sum_{i=1 \text{ to } m} \lambda_i \Phi_i (x)$

$\Phi_i (x) = 1$ if constraints i is violated

$= 0$ otherwise

Minimize $p(x) = \sum_{i=1 \text{ to } m} \lambda_i \mid \sum_{j=1 \text{ to } n} a(i,j)*x(j) - 1 \mid$

Timetabling Problems

- Assign times to a set of events
- Subject to constraints
- Most common constraint is an edge constraint between 2 events
  - Events e1 and e2 must not overlap in time
- Related kind of constraint is event-spread constraint
  - e.g. every student must have at least a 3-hr break between exams

Timetabling: Additional Constraints

- Assignment of rooms, places and times
  - Room capacities, travel times between locations
- Assignment of agents (instructors, tutors, invigilators) to events
  - Teaching loads, preferences, etc

Solving Timetabling Problems

- Find an acceptable timetable
  - All hard constraints are satisfied
- The inclusion of event-spread constraints, with all sorts of 'soft' constraints results in a very difficult optimization problem

Timetabling: Three Core Steps

- Decide how to represent the timetable as a chromosome
- Decide how to measure the ‘fitness’ of a timetable
- Decide on appropriate crossover and mutation operators

Style 1: Direct Representation

- A chromosome is a string of genes of total length *L*, where *L* is the number of events
- Each gene has a range of possible values – the possible start times (or time slots)
- "a b c d e f" represents a timetable in which
  - Event e1 starts at time a
  - Event e2 starts at time b, and so on.

Direct Representation

- Fitness function: a weighted sum of constraint violations
  - A candidate solution is tested against a long list of constraints, and penalties are assigned for each violation
- Evolution results in lower & lower weighted penalty scores until
  - (at least) timetables suffering no hard constraints appear
  - (soon after) timetables appear which suffer few or no violations of soft constraints
- Standard genetic operators
  - 1-pt, 2-pt and uniform crossover
  - Swapping sets of time assignments between parents
- Standard mutation
  - Altering the value of one or more time assignments in a single timetable

Style 2: Implicit Representation

- A chromosome is a permutation of events
- Interpretation makes use of some heuristic which helps decide where to schedule an event based on the partially-developed timetable so far
- "**e1 e3 e4 e5 e2**"
  - **Use heuristic H to schedule event e1**
  - **Use heuristic H to schedule event e3**
  - **Use heuristic H to schedule event e4, and so on**
- What is H?
  - **"Fits" events into slots by at least making sure all the hard constraints are satisfied**

Style 2: Implicit Representation

- Fitness function same as direct approach
  - Linear weighted sum of constraint violations
- Operators used must preserve the fact that the chromosome is a permutation
- More difficult to see what is a good schema
- Attractive feature of this approach
  - A smaller space is traversed
- Negative features
  - Chromosome interpretation becomes quite slow
  - Redundancy in the representation – distinct chromosomes may represent the same timetable

Modelling GA Problems for Optimization with Constraints

What Do Constraints Do?

- Limit the feasible portion of the search space
- Cause the feasible portion to be disjoint parts

4 Methods to Handle Constraints

1. Discard infeasible solutions
2. Preserve feasibility of solutions
3. Repair infeasible solutions to feasible ones
4. Penalize infeasible solution

1. Discard Infeasible Solutions

- **Discard infeasible solutions and try again**
  - Continue crossover & mutation until a feasible solution is produced
  - Limiting search to feasible region does not always enhance the search
- Advantage
  - There will never be infeasible solutions in the population
- Disadvantages
  - **Feasible solutions may be difficult to find**
    - When feasible search space is small
  - Spend much time in evaluation & rejection of infeasible solutions (especially when problem is highly constrained)
  - Considering no points outside the feasible regions, which can be bad

2. Preserve Feasibility of Solutions

- **Use problem-specific chromosome representation and/or genetic operators**
  - e.g. Heuristic GA: replace the random mechanism of crossover with deterministic or probabilistic mechanism based on heuristics
- Advantage
  - There will never be infeasible solutions in the population
- Disadvantages
  - Problem specific. Not all problems/constraints can be easily implemented this way.
  - Will improve performance for first few generations but performance will degrade rapidly as heuristic crossovers decrease genetic diversity
  - Feasible solutions may be difficult to find

3. Repair Infeasible Solutions

- Using special repair algorithms to “correct” any infeasible solutions so generated. Repair algorithms might be computationally intensive to run and the resulting algorithm must be tailored to the particular applications.
- The process of correcting a solution may be as difficult as solving the original problem.

3. Repair Infeasible Solutions

- Factors affecting performance
  - Quality of repair
  - Time to generate repaired solution
  - Diversity & nature of the mapping from infeasible to feasible solutions
- Advantage
  - Good for handling specific explicit constraints
- Disadvantages
  - Problem specific

4. Penalize infeasibility

- **Transform constrained optimisation into unconstrained optimisation**
  - Two kinds of penalty functions
    - Uniform
    - Use small penalties at the beginning and increase them gradually – to allow GA to explore more of the search space at the beginning
  - If penalty is high, more emphasis is placed on obtaining feasibility & GA will move quickly towards a feasible solution
  - If penalty is low, less emphasis is placed on feasibility & GA will never converge to a feasible solution

4. Penalize Infeasibility

How it works

- Generate the individuals in the population without considering the constraints and then penalize them by decreasing the goodness of the fitness function.
- Associate a penalty with all constraint violations. These penalties are factored into the fitness function in relation to the degree of constraint violation.

4. Penalize Infeasibility

- Advantage
  - It can consider infeasible solutions
- Disadvantages
  - **May never generate feasible solutions**
  - Initial small penalties may lead the GA to regions far from feasible solutions – regions where GA may be stranded in infeasible local optima