

Lecture 5: GAs for Grouping Problems

Constrained Optimisation Problems

- ▶ Many optimisation problems have inherent constraints on their solutions
- ▶ For such problems, not all solutions are valid
- ▶ We have already seen examples of this for permutation-based problems such as TSP
 - ▶ For ℓ objects, not all strings of length ℓ are valid solutions
 - ▶ Only those which are permutations of all ℓ objects are valid
 - ▶ The proportion of valid strings is hence $\frac{\ell!}{\ell^\ell}$
- ▶ Various approaches to these kinds of problems have been proposed, particularly
 - ▶ Penalties
 - ▶ Repair
 - ▶ Multiple objectives
 - ▶ Modified representations, operators and problem formulations

Penalties

- ▶ One solution to the generation of non-viable individuals is to implement some form of penalty
- ▶ The ultimate 'death-penalty' is to exclude them from the population
 - ▶ However, it is common to find the global optimum in the vicinity of a constraint boundary
 - ▶ Ignoring the degree of infeasibility of a solution discards some information
- ▶ Alternatively, we can modify the objective function by including a penalty function
 - ▶ The penalty function should be based on distance from a constraint boundary
 - ▶ E.g. number of constraints violated, and amount violated by
 - ▶ If penalties are too small, many non-viable solutions may spread in the population
 - ▶ If penalties are too large, the search will be too conservative and prevented from exploring the search space near its constraint boundaries

Repair

- ▶ Another strategy for dealing with non-viable offspring is to repair them before insertion into the population
- ▶ Repair operators will be problem and representation specific, but might include
 - ▶ Allele substitution to repair a 'fatal' mutation
 - ▶ Repair around a crossover site that has resulted in a non-viable solution
- ▶ We can either insert the original or repaired version of the offspring into the population
 - ▶ Inserting the repaired version may be too conservative by limiting search near a constraint boundary

Multiple Objectives

- ▶ We can also tackle constrained optimisation as a multiobjective optimisation problem
 - ▶ Maximise objective fitness
 - ▶ Minimize *unfitness* (infeasibility)
- ▶ Using a multiobjective GA we can find a non-dominated set of solutions with varying degrees of compromise between fitness and infeasibility
- ▶ This approach assumes that:
 - ▶ We can assign a meaningful objective fitness to infeasible solutions
 - ▶ We can meaningfully measure the extent to which the optimisation problem's constraints have been violated

Modified Representations, Operators and Formulations

- ▶ Good results are likely to be obtained by tailoring the GA to the problem at hand
- ▶ We can modify
 - ▶ Representations (encodings)
 - ▶ Genetic operators
 - ▶ The problem formulation itself
- ▶ One example of this was shown in lecture 2 for permutation problems, such as the TSP
 - ▶ Partially Matched Crossover (PMX)
- ▶ Let us now look at modified approaches to *grouping problems*

Grouping Problems

- ▶ Grouping problems are those in which objects must be assigned membership of different groups
 - ▶ E.g. the bin-packing problem (BPP)
 - ▶ For a set of objects having different weights, and a supply of bins of fixed size, find the assignment of objects to bins that minimises the number of bins needed
- ▶ Grouping problems can be *formulated* in two obvious ways
 - ▶ Object membership
 - ▶ Object permutation
- ▶ We will now look at drawbacks of these two approaches

Object Membership

- ▶ The object membership representation encodes each solution using one locus per object, and one allele per group
 - ▶ E.g.

AACBAB

- ▶ Encodes a solution with six objects assigned to three groups, A, B and C
- ▶ Such an encoding is clearly highly redundant
 - ▶ E.g. AACBAB and BBACBC both encode solutions where the same objects are assigned to the same groups
- ▶ It also suffers from *context insensitivity* under crossover

Context Insensitivity in Membership Encoding

- ▶ Object membership encoding disregards the *context* of group membership under standard crossover operators
- ▶ E.g. consider two-point crossover performed on the individuals

A|BC|ADD
and
C|AD|CBB

- ▶ These individuals encode identical solutions to the problem, in which objects 1 and 4, and 5 and 6 are grouped together
- ▶ Applying 2X at the marked crosspoints gives one of the offspring as

CBCCBB

- ▶ This individual differs completely from both its parents, despite the parents being identical to each other

Object Permutation

- ▶ Alternatively we can represent a solution to an object grouping problem as a permutation of the objects
- ▶ A heuristic is then used to decode the permutation
 - ▶ E.g. for each object in the chromosome, put it into the first bin in which it will fit, or in a new bin if no such bin exists
- ▶ As with membership encoding, permutation encoding is highly redundant
- ▶ E.g. the three individuals

3210|45678|9

0123|87654|9

87645|1032|9

- ▶ all encode the same solution

Context Insensitivity in Permutation Encoding

- ▶ Permutation encoding also suffers from context insensitivity
- ▶ As a solution is decoded from left to right, assignment of objects to groups depends on the objects that have appeared earlier in the chromosome
- ▶ Hence changing the objects encoded earlier in the chromosome may disrupt groups of objects encoded later in the chromosomes
- ▶ E.g. simply swapping the first and third bin contents in the encoded solution, where sufficient space remains in the third bin to fit object 3

012|3456|789

- ▶ results in the radically different solution

7893|4560|12

The Grouping Genetic Algorithm - Encoding

- ▶ The Grouping Genetic Algorithm (GGA) manipulates groups rather than individual objects
- ▶ The first major innovation is the encoding
- ▶ The standard object membership encoding is extended with a *group part*, based on one gene for one group, e.g.

ADBFEB:BEFDA
AAABBB:AB

- ▶ The group part identifies the groups present in the solution, the object part assigns individual objects into these groups
- ▶ As the number of groups is not normally predetermined in a grouping problem, chromosomes will be of variable length

The Grouping Genetic Algorithm - Operators

- ▶ The GGA operators work with the group part of the chromosome
- ▶ The group labels of the objects indicate which objects should be considered when manipulating a particular group
- ▶ Operators explicitly manipulate groups of objects, rather than independent objects
- ▶ Operators must work with variable length chromosomes

The Grouping Genetic Algorithm - Crossover

for both parents do

uniformly randomly select two crosspoints within group part of chromosome;

end

for 1 to 2 do

create clone offspring of first parent with same crosspoints;
inject first crossing section of second parent at first crosspoint of offspring;

for all objects occurring twice in offspring do

delete object from group in offspring in which object originally resided;

end

apply problem-dependent heuristics to satisfy constraints;
swap first and second parent roles;

end

The Grouping Genetic Algorithm - Operators

- ▶ Standard mutation is too destructive for grouping problems
- ▶ Hence a problem-specific mutation operator must be designed, which might
 - ▶ Create a new group
 - ▶ Eliminate an existing group
 - ▶ Shuffle objects among groups
- ▶ Inversion can be applied without modification to the group part of chromosomes in the GGA

Formae and Respect

- ▶ The GGA embodies the principles of *forma* and *respect*
- ▶ Forma
 - ▶ A subset of chromosomes that are similar in some, typically phenotypic, way
- ▶ Respect
 - ▶ Operators should *respect* the formae of the chromosomes on which they operate
 - ▶ Phenotypic characteristics should largely be transmitted intact
 - ▶ If parents share a particular characteristic, their offspring should inherit it
- ▶ These ideas capture a crucial aspect of GA design
 - ▶ Careful design of representation and operators, using knowledge of the problem domain
- ▶ Forma can be compared with the original explanation for how GAs work, schema theory, which we will cover later in the course