Artificial Intelligence Methods (G52AIM)

Dr Rong Qu

rxq@cs.nott.ac.uk

Miscellaneous & Implementation Issues

Local Search Algorithms

- Problem Specific Decisions
  - Besides some parameters in local search methods such as Simulated Annealing, Tabu Search, other decisions need to made

- Implementation issues
  - We also look at a list of issues that need to be considered when doing real implementations

Cost Function

- The evaluation function is calculated at every iteration
- Often the cost function is the most expensive part of the algorithm
  - This can be responsible for a large proportion of the execution time of the algorithm

Cost Function

- Therefore
  - We need to evaluate the cost function as efficiently as possible
  - Use Delta Evaluation
  - Use Partial Evaluation
  - Use approximations

Cost Function

- Delta Evaluation
  - Instead of evaluating every whole solution, as only small changes are being made between one solution and the next, it is possible to evaluate just the changes and update the previous cost function using the result of that calculation
  - Example:
    - 2-OPT – do not re-evaluate entire tour but just the change from the 4 edges involved

Cost Function

- Use cache*
  - The cache stores cost functions (partial and complete) that have already been evaluated
  - They can be retrieved from the cache rather than having to go through the evaluation function again and again

* Burke, et al. (1999)

Cost Function

- Fast approximation*
  - The evaluation function is approximated (one tenth of a second)
  - Potentially good solution are fully evaluated (three minutes)

* Rana, et al. (1996)

Cost Function

- If possible, the cost function should also be designed so that it can lead the search
  - One way of achieving this is to avoid cost functions where many states return the same value
    This can be seen as representing a plateau in the search space which the search has no knowledge about which way it should proceed
  - Bin Packing, graph colouring

Cost Function – Example

- Bin Packing
  - A number of items, a number of bins
  - Objective
    - As many items as possible
    - As less bins as possible
    - Other objectives depending on the problems

Cost Function – Example

- Bin Packing
  - Cost function?
    - a) number of bins
    - b) number of items
    - c) both a) and b)
  - How about there are weights for the items?

Cost Function

- Many cost functions cater for the fact that some solutions are illegal. This is typically achieved using constraints
  - Hard Constraints : these constraints cannot be violated in a feasible solution
  - Soft Constraints : these constraints should, ideally, not be violated but, if they are, the solution is still feasible
  - Examples: bin packing, timetabling (graph colouring)

Cost Function

- Hard constraints are given a large weighting. The solutions which violate those constraints have a high cost function value
- Soft constraints are weighted depending on their importance
- Weightings can be dynamically changed as the algorithm progresses. This allows hard constraints to be accepted at the start of the algorithm but rejected later

Neighbourhood

- How do you move from one state to another?
- When you are in a certain state, what other states are reachable?
  - Examples: bin packing, timetabling (graph colouring)

Neighbourhood

- Some results have shown that the neighbourhood structure should be symmetric, i.e. if you move from state *i* to state *j* then it must be possible to move from state *j* to state *i*
- **Every state must be *reachable* from every other**. Therefore, it is important, when thinking about your problem to ensure that this condition is met

Search space

- The smaller the search space, the easier the search will be
- If we define cost function such that infeasible solutions are accepted, the search space will be increased
- As well as keeping the search space small, also keep the neighbourhood small

Problem Specific Decisions

- Search space – small
  - size of neighbourhood
  - search is not restricted
- Cost function - easy to calculate
  - consider infeasible solutions
- Neighbourhood
- Overall aim
  - Make the most use of each iteration, whilst trying to ensure good quality solution

Implementation Issues

- Algorithm evaluation*
  - Optimality
  - Completeness
  - Time complexity
  - Space complexity

* See G51IAI

Implementation Issues

- Algorithm performance
  - Quality of the solution returned
  - Time taken by the algorithm
- We already have the problem of finding suitable SA parameters (cooling schedule)
- Besides the algorithm design, it is also usually very important that it is well implemented
  - This can take more work than the original algorithm

Improving Performance

- Initialisation
  - Start with a random solution and let the annealing process improve on that.
  - Might be better to start with a solution that has been heuristically built (e.g. for the TSP problem, start with a greedy search)

Improving Performance

- Hybridisation
  - *Memetic algorithms*
  - Combine two search algorithms
  - Relatively new research area

Improving Performance

- Hybridisation
  - Often a population based search strategy is used as the primary search mechanism and a local search mechanism is applied to move each individual to a local optimum
  - It may be possible to apply some heuristic to a solution in order to improve it

SA Modifications

- Acceptance probability
- Cooling schedule
- Cost function
- Neighborhood

SA - Acceptance Probability

- The probability of accepting a worse move is normally based on the physical analogy
- But is there any reason why a different function will not perform better for all, or at least certain, problems?

SA - Acceptance Probability

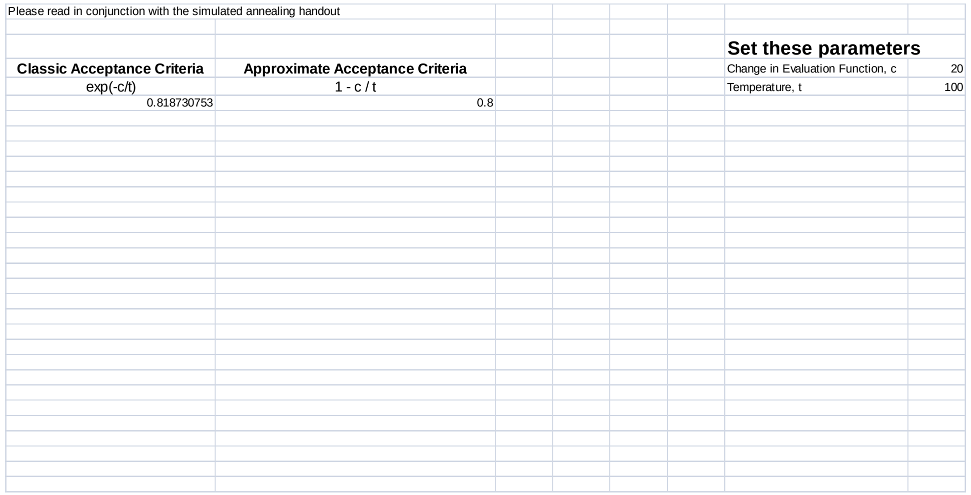
- Why should we use a different acceptance criteria?
  - The one proposed does not work
  - Or we suspect we might be able to produce better solutions
  - The exponential calculation is computationally expensive
  - Johnson (1991) found that the acceptance calculation took about one third of the computation time

SA - Acceptance Probability

- Johnson experimented with

$$P(\delta) = 1 - \delta/t$$

- This approximates the exponential



Please read in conjunction with the simulated annealing handout

Classic Acceptance Criteria	Approximate Acceptance Criteria	Set these parameters	
exp(-c/t)	1 - c / t	Change in Evaluation Function, c	20
0.818730753	0.8	Temperature, t	100

SA - Acceptance Probability

- A better approach was found by building a look-up table of a set of values over the range δ/t
- During the course of the algorithm δ/t was rounded to the nearest integer and this value was used to access the look-up table
- This method was found to speed up the algorithm by about a third with no significant effect on solution quality

SA - Cooling

- If you plot a typical cooling schedule you are likely to find that at high temperatures many solutions are accepted
- If you start at too high a temperature a random search is emulated and until the temperature cools sufficiently any solution can be reached and could have been used as a starting position

SA - Cooling

- At lower temperatures, a plot of the cooling schedule is likely to show that very few worse moves are accepted; almost making simulated annealing emulate hill climbing
- Taking this one stage further, we can say that simulated annealing does most of its work during the middle stages of the cooling schedule

SA - Cooling

- Connolly (1990) suggested annealing at a constant temperature

- But what temperature?
- It must be high enough to allow movement but not so low that the system is frozen
- But, the optimal temperature will vary
  - from one type of problem to another and also
  - from one instance of a problem to another instance of the same problem

SA - Cooling

- One solution to this problem is to spend some time searching for the optimal temperature and then stay at that temperature for the remainder of the algorithm
- The final temperature is chosen as the temperature that returns the best cost during the search phase

SA - Neighbourhood

- The neighbourhood of any move is normally the same throughout the algorithm, but…
- Could be changed as the algorithm progresses
  - For example, a different neighbourhood can be used to helping jumping from local optimal
  - Variable neighborhood search

SA - Cost Function

- Various researchers (e.g. Burke, 1999) have shown that the cost function can be responsible for a large proportion of the execution time of the algorithm
- Some techniques have been suggested which aim to solve this problem
  - Delta/partial evaluation
  - Approximation

SA - Cost Function

- Ross (1994) uses delta evaluation on the timetabling problem
  - Instead of evaluating every timetable as only small changes are being made between one timetable and the next, it is possible to evaluate just the changes and update the previous cost function using the result of that calculation

Summary

- Problem specific decisions
  - Cost function
  - Neighbourhood
  - Performance (initialisation, hybridisation)
- SA modifications

Learning Objectives

- Appreciation of the many choices to be made in designing real search algorithms
- Be able to use some of these in coursework

References

- E.K. Burke and G. Kendall, "Evaluation of Two Dimensional Bin Packing Problem using the No Fit Polygon", Proceedings of the 26th International Conference on Computers and Industrial Engineering, Melbourne, Australia, 15-17 December 1999, pp 286-291
- A. Rana, A.E. Howe, L.D. Whitley and K. Mathias. 1996. Comparing Heuristic, Evolutionary and Local Search Approaches to Scheduling. Third Artificial Intelligence Plannings Systems Conference (AIPS-96)
- Johnson, D.S., Aragon, C.R., McGeoch, L.A.M. and Schevon, C. 1991. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. Operations Research, 39, 378-406
- Connolly, D.T. 1990. An Improved Annealing Scheme for the QAP. EJOR, 46, 93-100
- P. Ross, D. Corne and F. Hsiao-Lan. 1994. Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation. In Y. Davidor, H-P Schwefel and R. Manner (eds) Parallel Problem Solving in Nature, Vol 3, Springer-Verlag, Berlin