Simulated Annealing and Tabu Search – Pseudocodes and Notes

1

Outline

- Simulated Annealing
- Tabu Search

2

Simulated Annealing Pseudocode

Algorithm below provides a pseudocode listing of the main Simulated Annealing algorithm for minimizing a cost function

Algorithm : Pseudocode for Simulated Annealing.

```
Input: ProblemSize, iterations_max, temp_max
Output: S_best
S_current ← CreateInitialSolution(ProblemSize);
S_best ← S_current;
for i = 1 to iterations_max do
    S_i ← CreateNeighborSolution(S_current);
    temp_curr ← CalculateTemperature(i, temp_max);
    if Cost(S_i) ≤ Cost(S_current) then
        S_current ← S_i;
        if Cost(S_i) ≤ Cost(S_best) then
            S_best ← S_i;
        end
    else if Exp( (Cost(S_current) − Cost(S_i)) / temp_curr ) > Rand() then
        S_current ← S_i;
    end
end
return S_best;
```

3

Simulated Annealing -1

- Simulated Annealing was designed for use with combinatorial optimization problems, although it has been adapted for continuous function optimization problems.
- The convergence proof suggests that with a long enough cooling period, the system will always converge to the global optimum.
  - The downside of this theoretical finding is that the number of samples taken for optimum convergence to occur on some problems may be more than a complete enumeration of the search space.

4

Simulated Annealing - 2

- Performance improvements can be given with the selection of a candidate move generation scheme (neighborhood) that is less likely to generate candidates of significantly higher cost.
- Restarting the cooling schedule using the best found solution so far can lead to an improved outcome on some problems.

5

Simulated Annealing - 3

- A common acceptance method is to always accept improving solutions and accept worse solutions with a probability of $P(accept) \leftarrow exp\left(\frac{e-e'}{T}\right)$,where T is the current temperature, e is the energy (or cost) of the current solution, and e' is the energy of a candidate solution being considered.
- The size of the neighborhood considered in generating candidate solutions may also change over time or be influenced by the temperature, starting initially broad and narrowing with the execution of the algorithm.
- A problem specific heuristic method can be used to provide the starting point for the search.

6

Tabu Search Pseudocode

for minimizing a cost function; simple Tabu Search algorithm with short term memory, without intermediate and long term memory management.

Algorithm : Pseudocode for Tabu Search.

```
   Input: TabuList_size
   Output: S_best
 1 S_best ← ConstructInitialSolution();
 2 TabuList ← ∅;
 3 while ¬ StopCondition() do
 4 |   CandidateList ← ∅;
 5 |   for S_candidate ∈ Sbest_neighborhood do
 6 |   |   if ¬ ContainsAnyFeatures(S_candidate, TabuList) then
 7 |   |   |   CandidateList ← S_candidate;
 8 |   |   end
 9 |   end
10 |   S_candidate ← LocateBestCandidate(CandidateList);
11 |   if Cost(S_candidate) ≤ Cost(S_best) then
12 |   |   S_best ← S_candidate;
13 |   |   TabuList ← FeatureDifferences(S_candidate, S_best);
14 |   |   while TabuList > TabuList_size do
15 |   |   |   DeleteFeature(TabuList);
16 |   |   end
17 |   end
18 end
19 return S_best;
```

7

Tabu Search - 1

- Tabu search was designed to manage an embedded hill climbing heuristic, although may be adapted to manage any neighborhood exploration heuristic.
- Tabu search was designed for, and has predominantly been applied to discrete domains such as combinatorial optimization problems.
- Candidates for neighboring moves can be generated deterministically for the entire neighborhood or the neighborhood can be stochastically sampled to a fixed size, trading off efficiency for accuracy.

8

Tabu Search - 2

- Intermediate-term memory structures can be introduced (complementing the short-term memory) to focus the search on promising areas of the search space (intensification), called aspiration criteria.
- Long-term memory structures can be introduced (complementing the short-term memory) to encourage useful exploration of the broader search space, called diversification. Strategies may include generating solutions with rarely used components and biasing the generation away from the most commonly used solution components.