Computational Intelligence

Unit # 10

Tabu Search

- Tabu search uses common-sense ideas to enable the search process to escape from a local optimum.
- It uses a local serach procedure to find a local optimum.
- A key strategy of tabu search is that it then continues the search by allowing *non-improving moves* to the best solutions in the neighborhood of the local optimum.
- The danger with this approach is that after moving away from a local optimum, the process will cycle right back on the same local optimum.

Tabu List

- To avoid this, a tabu search temporarily forbids moves that would return to (or perhaps towards) a solution recently visited.
- A **tabu list** records these forbidden moves, which are referred to as tabu moves.
- The only exception to forbidding such a move is if it is found that a tabu move actually is better than the best feasible solution found so far.
- This use of *memory* to guide the search by using tabu lists to record some of the recent history of the search is a distinctive feature of tabu search.

Sub-Tour Reversal in TSP

- A sub-tour reversal adjusts the sequence of cities visited in the current trial solution by selecting a subsequence of the cities and simply reversing the order in which that subsequences of cities is visited.
- The subsequence being reversed can consist of as few as two cities, but also can have more.
- The sub-tour reversal is an example of a local improvement procedure.

Outline of a Basic Tabu Search Algorithm

- Initialization
  - Start with a feasible initial trial solution.
- Iteration
  - Use an appropriate local search procedure to define the feasible moves into the local neighborhood of the current trial solution.
  - Eliminate from consideration any move on the current tabu list unless that move would result in a better solution than the best trial solution found so far.
  - Determine which of the remaining moves provides the best solution and adapt it regardless of whether it is better or worse than the current trial solution.
  - Update the tabu list. If the tabu list is already full, delete the oldest member of the tabu list.
- Stopping Rule
  - Same as used for other algorithms discussed in this course

Example Revisited

	A	B	C	D	E	F	G
A	0	12	10				12
B	12		8	12			
C	10	8		11	3		9
D		12	11		11	10	
E			3	11		6	7
F				10	6		9
G	12		9		7	9	

A – B – C – D – E – F – G - A : 12 + 8 + 11 + 11 + 6 + 9 + 12 = 69

Working of Tabu Search

- Initial solution
  - A – B – C – D – E – F – G – A : Distance = 69
  - Tabu list: Blank at this point
- Iteration 1: Reverse C – D
  - Deleted Links: B – C, D – E
  - Added Links: B – D, C – E
  - Tabu List: B – D, C – E
  - A – B – D – C – E – F – G – A : Distance = 65
- Iteration 2: Reverse C – E – F
  - Deleted Links: D – C, F – G
  - Added Links: D – F, C – G
  - Tabu List: B – D, C – E, D – F, C – G
  - A – B – D – F – E – C – G – A : Distance = 64

Working of Tabu Search (Cont'd)

- Iteration 3: Reverse C – G
  - Deleted Links: E – C, G – A
  - Added Links: E – G, C – A
  - Tabu List: ~~B – D, C – E~~, D – F, C – G, E – G, C – A
  - A – B – D – F – E – C – G – A : Distance = 66
- The process continues in a similar fashion until one of the stopping condition is satisfied.

Simulated Annealing

- Developed in 1983, SA is another widely used metaheuristic that enables the search process to escape from a local optimum.
- The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.
- The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

Analogy

- By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends both on the difference between the corresponding function values and also on a global parameter *T* (called the *temperature*), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when *T* is large, but increasingly "downhill" as *T* goes to zero.

Working of the Algorithm

- At each step, the SA heuristic considers some neighboring state *s′* of the current state *s*, and probabilistically decides between moving the system to state *s′* or staying in state *s*.
- These probabilities ultimately lead the system to move to states of lower energy.
- Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.

Method

- SA's major advantage over other methods is its ability to avoid becoming trapped in local minima.
- The algorithm employs a random search which not only accepts changes that decrease the objective function f, but also some changes that increase it.
- The latter are accepted with a probability
  - $P = \exp(\Delta f/T)$
  - Where Δf is the increase in f and T is a control parameter "temperate".

Outline of Simulated Annealing Algorithm

- Initialization
  - Start with a feasible initial trial solution
- Iteration
  - Use the *selection rule* to select the next trial solution.
  - If none of the immediate neighbors of the current trial solution are accepted, the algorithm is terminated.
- Check the temperature schedule
  - When the desired number of iterations have been performed at the current value of T, decrease T to the next value in the temperature schedule and resume performing iterations at this next value.
- Stopping Rule
  - Same as used for other algorithms discussed in this course

TSP Example: Temperature Schedule

- Five iterations are performed at each of the five values of T (T_1, T_2, T_3, T_4, T_5) in turn, where
  - $T_1 = 0.2Z_c$ when Z_c is the objective function value for the initial trial solution
  - $T_2 = 0.5T_1$
  - $T_3 = 0.5T_2$
  - $T_4 = 0.5T_3$
  - $T_5 = 0.5T_4$

TSP Example: Working

- Initial Trial
  - A – B – C – D – E – F – G – A, Zc = 69, T1=0.2xZc = 13.8
- Second Iteration
  - Pick two random numbers to find the swapping slots. Suppose the sub-tour of cities C – D is reversed.
  - A – B – D – C – E – F – G – A, Zn = 65
- Third Iteration
  - Reverse C – E – F
  - A – B – D – F – E – C – G – A, Zn = 64
- Fourth Iteration
  - Reverse C – G
  - A – B – D – F – E – G – C – A, Zn = 66 and Zc = 64 implies $\Delta f = -2$
  - Prob {acceptance} = exp (-2/13.8) = 0.865
  - If the next random number generated is less than 0.865, this candidate solution will be accepted as the next trial solution. Otherwise it will be rejected.

Differential Evolution

- Differential evolution (DE) is a population-based search algorithm.
- The algorithm draws inspiration from the field of evolutionary computation, as it embeds implicit concepts of mutation, recombination and fitness-based selection to evolve good solutions to a problem of interest by manipulating a population of solution encodings.
- An individual in DE is generally comprised of a real-valued chromosome.

Working of DE

- At the start of the algorithm, a population of N, d-dimensional vectors $X_j = (x_{i1}, x_{i2}, \ldots, x_{id})$, $j = 1, \ldots, N$, each of which encode a solution, is randomly initialized and evaluated using a fitness function f.
- During the search process, each individual (j) is iteratively refined.

Three Major Steps

- The modification process has three steps:
  - Create a variant vector which encodes a solution, using randomly selected members of the population (mutation step).
  - Create a trial vector, by combining the variant vector with j (crossover step).
  - Perform a selection process to determine whether the newly-created trial vector replaces j in the population.

Mutation

- Under the mutation operator, for each vector $X_j(t)$ a variant vector $V_j(t+1)$ is obtained:
  - $V_j(t + 1) = X_m(t) + F(X_k(t) - X_l(t))$
  - where k, l,m ∈ 1, . . .,N are randomly selected indices, and all the indices ≠ j (X_m is referred to as the base vector, and $X_k(t) - X_l(t)$ is referred to as a difference vector).

Crossover

- Selecting the three indices randomly implies that all members of the current population have the same chance of being selected, and therefore influencing the creation of the difference vector.
- The difference between vectors X_k and X_l is multiplied by a scaling parameter F (typically F ∈ (0, 2]).
- The scaling factor controls the amplification of the difference between X_k and X_l, and is used to avoid stagnation of the search process.

Self-Scaling Mutation

- A notable attribute of the mutation step in DE is that it is self-scaling.
- The size/rate of mutation along each dimension stems solely from the location of the particles in the current population.
- The mutation step self-adapts as the population converges leading to a finer-grained search.
- In contrast, the mutation process in the canonical GA is typically based on draws from a separately defined (fixed) probability density function.

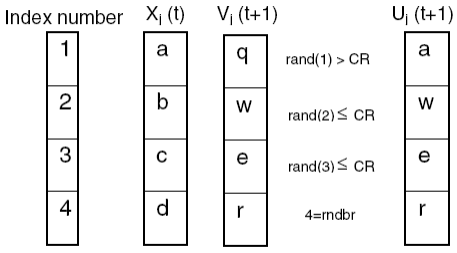
Cross Over

- Following the creation of the variant vector, a trial vector $U_j(t + 1)$ = (uj1, uj2, . . . , ujd) is obtained:

$$U_{jk}(t+1) = \begin{cases} V_{jk}(t+1), \text{ if } (rand \leq CR) \text{ or } (j = rnbr(ind)) \ ; \\ X_{jk}(t), \text{ if } (rand > CR) \text{ and } (j \neq rnbr(ind)). \end{cases}$$

- where k = 1, 2, . . . , d, rand is a random number generated in the range (0,1), CR is the user-specified crossover constant from the range (0,1), and rnbr(ind) is a randomly chosen index chosen from the range (1, 2, . . . , d).
- The random index is used to ensure that the trial solution differs by at least one component from $X_j(t)$.

Cross Over (Cont'd)



Mutation and Cross Over

Parents

a	b	c
4.435	1.222	9.735
6.567	3.986	1.503
2.456	2.325	7.908
4.567	1.254	3.420
3.976	5.124	5.394

i=1

First selection (i_4)

4.567	1.254	3.420

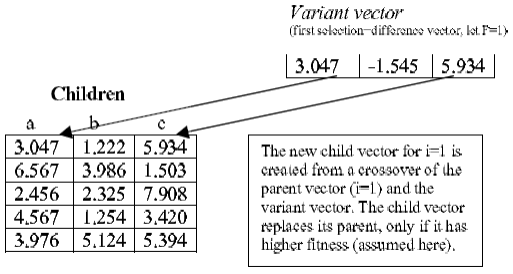
Second and third selections (i_3 & i_5)

3.976	5.124	5.394
2.456	2.325	7.908

Difference vector (i_5-i_3)

-1.520	-2.799	2.514

Mutation and Cross Over (Cont'd)



- The resulting trial (child) solution replaces its parent if it has higher fitness (a form of selection), otherwise the parent survives unchanged into the next iteration of the algorithm

Summary of DE

- Initialize Population
- Evaluate fitness of population members
- Do
  - For each member of the population
    - Perform mutation operator by creating a variant vector with the help of three randomly selected individuals
    - Perform crossover operator by combining the variant vector with the individual. The resultant vector is called trial vector
    - Using a selection mechanism (Binary Tournament, etc.), decide if the trial vector replaces the original individual or not.
  - Next
- While (Not Finished)