Metaheuristic Algorithms.

Lab 2: Combinatorial optimization problems.
Trajectory based local and global search:
Simulated Annealing
Tabu Search

---

1. Combinatorial optimization problems.

The search space of combinatorial optimization problems is usually finite but of large size. Thus an exhaustive search space exploration is not inapplicable.

Two well-known combinatorial optimization problems, which have also Două dintre cele mai cunoscute problem de optimizare combinatorială, care au și o serie de aplicații practice, sunt:

- Problema comis-voiajorului (travelling salesman problem)
- Problema rucsacului (knapsack problem)

1. 1. Problema comis-voiajorului (Travelling Salesman Problem).

TSP is a well known combinatorial optimization problem asking to find the optimal route for a salesman who has to visit a set of n towns. It is a constrained optimization problem characterized by:

- Constraint: the salesman visits each town exactly once
- Objective function: the cost of the tour should be minimized

The classical TSP is equivalent with the problem of finding an optimal Hamiltonian tour in a complete graph. TSP can be solved exactly for small values of n but, since the number of possible tours is (n-1)!/2, for large values of n there are no efficient exact methods. TSP belongs to the class of NP-complete problems.

TSP is important not only from a theoretical point of view but also from a practical point of view since there are several real-world problems which can be formulated as a TSP:

- Vehicle Routing Problem (VRP): find the optimal route for vehicles
- Control of drilling machines which are used to construct boards for integrated circuits
- Find shortest routes through selections of airports in the world
- Reconstruct DNA sequences starting from subsequences

Other applications are listed at [http://www.tsp.gatech.edu/apps/index.html]

TSP has several variants:

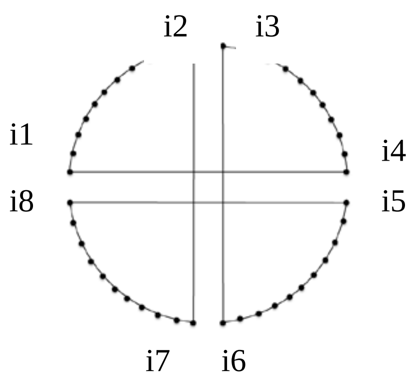
- *Asymmetric TSP*: the passing from one node to another one depends on the tour orientation.
- *Sequential Ordering Problem* – SOP: there are additional constraints specifying that a given node should be visited before another one.

- *Capacitated vehicle routing problem* – CVRP: find optimal tours for a set of trucks which have to transport products from a warehouse to different customers. The trucks have all the same capacity.
- *Generalized TSP*: the nodes correspond to clusters of locations and there are several arcs between nodes.

Besides exact methods there exist a lot of heuristic methods based on incremental improvements of the current tour. One of the most used heuristics for TSP is the Lin-Kernighan heuristic which is based on replacing some arcs of the current tour with other ones such that the total cost becomes smaller. The simplest case is when just two arcs are replaced (2-opt transformation) which is equivalent to reversing the order of visiting the nodes belonging to a subtour.

Example: Let us consiuder 6 nodes: A,B,C,D,E,F . If the current tour is (A,C,B,E,F,D), by replacing the arc (A,C) with the arc (A,F), and the arc (F,D) with (C,D) and by reversing the order of visiting the nodes B and E one obtains the tour (A,F,E,B,C,D). It is easy to see that this transformation can be obtained by reversing the subtour (C,B,E,F).

Another perturbation heuristic for TSP is that based on 4 interchanges (double bridge transform) which transforms a route [i1..i2][i3..i4][i5..i6][i7..i8] into [i2..i1][i4..i3][i6..i5][i8..i7].



1.2. Knapsack Problem

The classical variant of the knapsack problem is: "Let us consider a set of n objects, each one being characterized by a given weight and a given value. Select a subset of objects such that the total size of the selected objects is smaller than a knapsack capacity and the total value of the selected objects is as large as possible."

The search space is represented by all possible subsets of the set of n objects, thus the serach space size is 2^n.

Real world problems which can be formulated as the knapsack problem are:

- Financial portfolios construction (the aim being the maximization of the profit such that the amount of investment is lower than a given threshold).
- Resource allocation (the selection of some tasks which can use a given resource such that the resource is not overloaded and some gain is maximized).
- The selection of some products to be placed in a container or warehouse.

Variants of the problem:

- Multi-criterial case: the aim is not only to maximize a value but optimize several criteria
- Multi-dimensional case: the “size”/”weight” of an object is not specified by a single value but by multiple values
- Multiple knapsacks: several knapsacks are used (this is related to the bin packing problem)

2. Simulated Annealing (SA)

2.1 **Method description**

Simulated Annealing is a metaheuristic characterized by the fact that lower quality configurations may be accepted. The decision on the acceptance of such configurations is taken probabilistically, and the acceptance probability depends on a parameter called “temperature” (by analogy with the temperature of physical systems which are involved in a thermal process, e.g. annealing of alloys). The probability of accepting a lower quality configuration is higher if the temperature is higher.

General structure of Simulated Annealing:

```
S=initial configuration
T=initial value of the temperature
Repeat
  S’=perturb(S)
  If accept(S,S’,T) then S=S’
  T=perturb(T)
Until <stopping condition>
```

The perturbation depends on the problem to be solved and the probability to accept the transition form a configuration S to a configuration S’ depends on the loss of quality (if the loss is small the acceptance probability is higher). An example of the implementation of an acceptance rule is (in the case of a minimization problem):

```
Accept(S,S’,T)
    If rand(0,1)<exp((f(s)-f(s’))/T) then
      Return True
    Else
      Return False
```

2.2. Solving TSP using Simulated Annealing

In order to solve a problem by using Simulated Annealing there are several elements to be chosen:

- *Solution encoding.* The natural encoding variant for TSP is the permutation: a tour through n nodes can be described as a permutation of order n. This encoding ensures the satisfaction of the constraint of visiting only once each node.

Example: If the towns are numbered as follows: 1-A, 2-B, 3-C, 4-D, 5-E, 6-F then the route (A,C,B,E,F,D) corresponds to the permutation (1,3,2,5,6,4)

- *Local search mechanism (construction of a new configuration starting from the existing one by perturbation)* The simplest mechanism is based on a 2-opt transformation:
    - Choose two random indices i and j such that 1<=i<j<=n
    - Reverse the order of elements in the permutation having indices between i and j.

  Example: If the current tour is described by the permutation (1,3,2,5,6,4) and i=2, j=5 then the new permutation will be: (1,6,5,2,3,4)

- *Acceptance probability.* The probability to accept a configuration S' obtained from the configuration S can be computed by using the Boltzmann distribution:

  P(S'|S)=min{1,exp(-(cost(S')-cost(S))/T(k)}

  where cost(S) is the cost of configuration S and T(k) is a control parameter (temperature).

- *Cooling schedule.* If T(k) denotes the temperature corresponding to the iteration k then the value corresponding to the next iteration can be computed as follows:

    - *T(k+1)=T(0)/log(k+c),* c being a constant
    - *T(k+1)=T(0)/k*
    - *T(k+1)=a T(k),* with *a* denoting a value smaller but close to 1 (e.g. a=0.99)

  Remark. The initial value of the temperature (T(0)) should be large enough to allow the transition between any two configurations.

- *Stopping condition.* The stopping criterion can be related to the value of the temperature (stop when the temperature is low enough), to the number of iterations (stop after a given number of iterations) or to the value of the objective function (cost of the tour).

Application 2. Implement the Simulated Annealing for TSP. Use test instances from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Hint. A variant is implemented in SA_TSP.m.

Call: SA_TSP(10000,0.001)

Exercises:

1. Write a Scilab function to read data from *.tsp files (downloaded from TSPLib) and analyze the behavior of the algorithm for the problems: eil51.tsp, eil76.tsp, eil101.tsp
2. Test the algorithm for each of the cooling schedules mentioned above.
3. Modify the function which compute the cost of a tour such that the distance between two nodes is computed only once (hint: store the distances in a matrix)

2.3 Solving the knapsack problem using Simulated Annealing

a) *Solution encoding*: binary vector
   - s_i=1 if object i is selected
   - s_i=1 if object i is not selected

b) *Local perturbation*: change the value of a randomly selected component: s_i= s_i -1

c) *Evaluation of a configuration*: a common variant is to include in the objection function the degree of constraints satisfaction (penalty function technique) – the value of a configuration which do not satisfy the constraint is penalized by a term which is proportional with the amount by which the constraint is violated (e.g. the weight which overpasses the knapsack capacity).

$$V(s)=\begin{cases}\sum_{i=1}^{n} v_i s_i & \text{if } \sum_{i=1}^{n} w_i s_i \le c\\ \lambda\sum_{i=1}^{n} v_i s_i+(1-\lambda)(c-\sum_{i=1}^{n} w_i s_i) & \text{if } \sum_{i=1}^{n} w_i s_i > c\end{cases}$$

The parameter λ belongs to (0,1) and allows the control of the relative importance of the constraints with respect to the optimization criterion.

Application 2. Implement a Simulated Annealing algorithm for a knapsack problem.

Hint. An implementation variant is described in SA_knapsack.

3. Tabu Search

3.1. Method description

Tabu Search is a metaheuristics which uses an iterated local search which relies on the usage of a list of already visited configurations which become "forbidden" (tabu) at least for a given number of iterations.

General structure of Tabu Search:

```
S=initial configuration
TabuList=[]   // the tabu list is initially empty
Iter=1
Repeat
  S'=perturb(S,TabuList)
  If better(S',S) then S=S' endif
    Iter=iter+1
Until iter<=iterMax
```

The perturbation of the current configuration is based on the identification (in its neighborhood) of a better configuration which is not in the tabu list. Once a configuration is chosen it is inserted in the tabu list. The tabu list is implemented as a circular queue (when the maximal size of the list is reached the first element in the list is removed).

```
Perturb(S,TabuList)
Sbest=S
For <for each element  S’ from the neighborhood N(S)>
  If  better(S’,Sbest) and <S’ is not in TabuList> then
     Sbest=S
  Endif
Endfor
S=Sbest
<update the tabu list by adding S>
```

The function better(S’,S) checks if configuration S’ is better than configuration S. Unlike Simulated Annealing which uses directly the value of the objective function to compute the acceptance probability, in Tabu Search it is enough to decide which of the configurations is better. This means that the constraints can be analyzed directly, without using the penalty method.

3.2. Solving the knapsack problem using Tabu Search

a) *Solution encoding*: binary vector
b) *Local perturbation*: change the value (0->1,1->0) of a randomly selected component
c) *Tabu list structure*: it contains candidat solutions (binary vectors)
d) *Comparison between two candidate solutions*:
   - If both S and S’ are feasible then the configuration having a higher value is better.
   - If only one of the solutions is feasible then it is better than the other one (a feasible solution is always better than an unfeasible one).
   - If none of the solutions is feasible then that which violates less the constraint is better

Application 3. Implement a Tabu Search algorithm to solve the knapsack problem.

Hint: an implementation variant is in TS_Knapsack.sci

Appendix: random values in SciLab

Values corresponding to random variables with various distributions can be generated using the grand function

This function returns a mxn matrix of random values. The typical call is:

```
grand(m,n,”<distribution type>”, <distribution parameters>)
```

Examples:

1. Uniformly distributed integer random values belonging to {inf,inf+1,inf+2,…,sup}

   ```
   grand(m, n, "uin", inf, sup)
   ```

2. Uniformly distributed real values belonging to [inf,sup)

   ```
   grand(m, n, "unf", inf, sup)
   ```

3. Random values corresponding to a normal distribution with mean m and standard deviation s

   ```
   grand(m, n, "nor", m, s)
   ```

Another useful function is that which generates n random permutations of a vector (vect):

```
grand(n, "prm", vect)
```