The Application of Natural Computation for Solving the Traveling Salesman Problems

Jiang Huowen[1,a] Xiong huanliang[2,b] Zhang Huiyun[3,c]

[1] Collage of Mathematics & Computer Science, Jiangxi Science & Technology Normal University, Nanchang 330038, China

[2] Software College, Jiangxi Agricultural University,Nanchang 330045, China

[3]Jiangxi Water Resources Institute,Nanchang 330045, China

[a]Jhw_604@163.com [b]xionghuanliang@126.com [c] ashzhhy@126.com

Keywords: Traveling Salesman Problem, natural Computation, Genetic Algorithm, Continuous Hopfield Algorithm, Ant Colony Optimization Algorithm

Abstract. Traveling Salesman Problems is a classical kind of combined optimization problems. Solving TSP effectively can produce very important theoretical value in computable theory, and also has very high applied value in practice. Solving travel salesman problems by using some traditional algorithms is limited, The article discuss several natural computation methods of solving travel salesman problems, including solution of genetic algorithm, solution of continuous Hopfield algorithm and solution of Ant colony optimization algorithm.

Introduction

The travelling salesman problems (TSPs for short) asks the following question: Given a list of cities and the distances between each pair of cities, how can a salesman travel along the shortest possible route that visits each city exactly once and returns to the origin city? The TSP can be formulated as the Hamiltonian circuit problem in graph theory which can be described mathematically as: Assume D= (d_{ij}) is the distance matrix of a graph, where d_{ij} stores the distance between vertex i and vertex j contained in the graph, then TSPs seeks for the shortest cycle that visits each vertex exactly once. The TSPs can be applied to traffic managing, network routing, chip designing and etc., which makes solving it of practical value. Traditional approaches, such as exhaustive search, greedy algorithm, dynamic programming and etc., are all faced with the same problem: with the increase of N (the number of listed cities), the search space of the TSPs grow exponentially, yielding massive amount of computation and intolerable running time[1]. Natural computation algorithms are based on adopting some natural principles , guide and optimize the solution space searching. Natural computation algorithms have characters such as high parallelism, self-organization, self-adaptation, self-learning and etc., which make them viable to solve highly complex non-linear problems that traditional solvers can't conquer[2]. Three typical natural computation algorithms are applied to solving TSPs in this paper.

Genetic algorithm for solving TSPs

Genetic algorithm, inspired by Darwin's evolutionary theory, searches solution space effectively for the best solution that satisfies predefined stopping criterion(commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population) by generating child generation populations through genetic operations like crossover and mutation. Genetic algorithm, inspired by Darwin's evolutionary theory, searches solution space effectively for the best solution that satisfies predefined stopping criterion(commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population) by generating

child generation populations through genetic operations like crossover and mutation[3].First of all, we need a encoding scheme to let chromosomes represent permutations of all cities in play. Let's suppose that n cities are to be visited, then a possible route can be represented by a integer vector of size n: $i_1, i_2, \mathbf{K}, i_n$, where i_m $(1 \le m \le n)$stands for the existence of city m in the route and the vector holds a full permutation of the integer numbers ranging from 1 to n, that is, each number within the range appears exactly once in the vector's elements. Detailed steps of the genetic algorithm for solving the TSPs are as follows:

(1)Initialize P(0), the initial population, with a population size of N and let t=0;

(2) Select u parents from P(t), the t-th generation population, randomly;

(3) Generate u children of bred by the u parents selected in Step (2) through the genetic operator of crossover;

(4) Select two individuals from P(t) randomly. Replace one of them with the fittest child generated in Step (3) and, after applying the genetic operator of mutation on the rest u-1 children, replace the other one with the fittest individual of the u-1 mutated children;

(5)Repeat Step (2) to Step (4) until all individuals in P(t) are replaced with N newly generated individuals.

(6)If $t<T$ (the maximum number of generations), then $t=t+1$ and go to Step (2); If not, regard the fittest individual in the current population as the best solution.

Continuous hopfield network for solving TSPs

In essence, the TSPs is a combinatorial optimization problem which aims at finding an optimal object from a finite set of objects. For general combinatorial optimization problem, let $F = \{x \in D \mid g(x) \ge 0\}$ denote the feasible region, where D is a finite set composed of limited decision variables, $x \in F$ denote x being a candidate solution, $f : D \to R$ denote the objective function, then the candidate solution x^* is said to be an optimal solution for the combinatorial optimization problem (D, F, f) if there exists $f(x^*) = \min\{f(x) \mid x \in F\}$. (D, F, f) here denotes $\min f(x)$, $(s.t.\ g(x) \ge 0,\ x \in D.\ )$ For a combinatorial optimization problem like TSPs, hopfiled network can transform the objective function into the energy function of the network, mapping solutions to the problem onto states of the network[4]. An optimal solution for the problem is obtained when the network eventually converge to a state that is a local minimum in the energy function. The detailed procedures of using a Hopfield network to solve the TSP are as follows:

(1) Map states of the neural network onto solutions to the TSP. Construct a neural network of size n×n, where the state of a neuron represents where a certain city appears in a certain possible route. To be more exact, let V_{xi} denote the state of neuron xi, where x($x \in$ {1,2,…, n}) represents the appearance of C_x(the x-th city of all listed cities) in the route while i represents that C_x is the i-th city to be visited in the route; if V_{xi}=0, then C_x doesn't appear at the i-th position in the route, which means other city occupies the i-th position. Hence it can be seen that matrix V of size $n \times n$ can well store the sequence of all listed cities to be visited which leads to a possible route for the TSPs.

(2) Define the energy function of the neural network. The energy function is defined as: $E=E_1+E_2+E_3+E_4$, where $E_1=\frac{A}{2}\sum_x\sum_i\sum_{j\neq i} v_{xi}v_{xj}$ (A is a constant and A>0) and it is guaranteed that, when the number of 1s in each row of matrix V is not greater than 1(which implies that each city is visited exactly once), $E_{1\min}=0$; $E_2=\frac{B}{2}\sum_i\sum_x\sum_{y\neq x} v_{xi}v_{yi}$ (B is constant and B>0) and it is guaranteed that, when the number of 1s in each row of matrix V is not greater than 1, $E_{2\min}=0$; $E_3=\frac{C}{2}(\sum_x\sum_i v_{xi}-n)^2$ (C is constant and C>0) and it is guaranteed that, when the number of 1s in each row of matrix V is n,

E_{3min}=0; $E_4=\frac{D}{2}\sum_x\sum_{y\neq x}\sum_i d_{xy}v_{xi}(v_{y,i+1}+v_{y,i-1})$ (D is constant and D>0) and the definition E_4 contains distance information of valid paths.

(3) With energy function determined, construct the neural network by backstepping. Compare our energy function with standardized energy function and we have the connection weight over the edge connecting neuron x_i and neuron y_j:

$T_{xi,yj}=-Ad_{xy}(1-d_{ij})-Bd_{ij}(1-d_{xy})-C-Dd_{xy}(d_{j,j+1}+d_{j,j-1})(1-d_{xy})$ where d_{ij},d_{xy} was defined as $d_{ij}=\begin{cases}1, i=j\\ 0, other\end{cases}$, Input bias $I_{xi}=Cn$.

(4) Setup and run the network. We will obtain an optimal solution when the network converges. Noting that the formula for calculating $T_{xi,yj}$ is given above, we have the running equation of the network as follows:

$$C_{xi}\frac{dU_{xi}}{dt}=-\frac{U_{xi}}{R_{xi}}-A\sum_{j\neq i}v_{xj}-B\sum_{y\neq x}v_{yi}-C(\sum_x\sum_i v_{xi}-n)$$
$$-D\sum_{y\neq x}d_{xy}(v_{y,i+1}+v_{y,i-1})$$

Let $v_{xi}=g(u_{xt})$, assign proper value to (A, B, C, D) and U_0 (which is the initial value), and run the step iteratively until convergence occurs. In the TSPs, v_{xi} is supposed to be either 0 or 1, but due to the continuity of hopfield network, v_{xi} changes continuously within [0,1] in practice. In the revolution process, a minority of neurons output value in an increasing manner while the rest does otherwise.

Ant colony optimization algorithm for solving TSPs

The ant colony optimization algorithm learns from the foraging behavior of ants, searching for the global best solution heuristically under the guidance of pheromone until convergence occurs[5]. Research shows: ants are always able to find the best path between their colony and food source, because the more ants go along a certain path, the more pheromone is laid on the path, resulting in a better chance of other ants travelling along the path and reinforce it, which eventually leads to the convergence of all ants picking the best path. Based on the foraging process of ants, procedure of the algorithm is demonstrated simply as follows: Place m ants randomly at n listed cities. For the first round of visit, each ant decides which neighboring city to visit next according a strategy of roulette gaming and visits the selected city. Each ant goes on its journey until all cities are visited. Move the initial starting city for each ant to the tail of its visit sequence. Thus far, the first round of visit is over and update pheromone laid on visited paths. Repeat visiting for rounds until the maximum times of iteration is reached or there exist two solutions that are extremely close to each other.

Assume that there are n cities and m ants in play. Let d_{ij}=(i,j=1,2,... ,n) denote the distance between city i and city j, $t_{ij}(t)$ denote the intensity of pheromone left on the path between city i and city j. Each ant's decision of which neighboring city to visit next is based on the intensity of pheromone on optional paths , Letting $P_{ij}^k(t)$ represent the probability of ant k moving from city i to city j, we have: Letting $P_{ij}^k(t)$ represent the probability of ant k moving from city i to city j, we have:

$$P_{ij}^k(t)=\begin{cases}\dfrac{[t_{ij}(t)]^a[h_{ij}(t)]^b}{\sum_{s\in J_k(i)}[t_{is}(t)]^a[h_{ij}(t)]^b}, & j\in J_k(i)\\ 0\end{cases} \quad \cdots ①$$

where $k(i)= \{1,2,... ,n\}$-$tabu_k$ is the set of cites ant k can go to in the next step . The set $tabu_k$ is the tabu list of ant k which records all the cities that have been visited by the ant k so far. Ant k is said to finish a round tour, the path of which is a candidate solution to TSPs, if all of the n listed cities are contained in $tabu_k$ (which implies that ant k traversed all n cities) and ant k returns to the city where its trip started. h_{ij}, called a heuristic factor or visibility, represents the level of expectation of ant k moving from city i to city j, usually taking the value of the reciprocal of d_{ij}. a and b respectively denote the relative importance of pheromone and that of heuristic factors. Pheromone on each path is globally updated according to Equation ② below after all ants finished its round trip.

$$t_{ij}(t+1) = (1-r)*t_{ij}(t) + \Delta t_{ij} \qquad \ldots ②$$

$$\Delta t_{ij} = \sum_{k=1}^{m} \Delta t_{ij}^{k} \qquad \ldots ③$$

where $r(0 < r < 1)$ denotes the volatility coefficient of pheromone; 1- r is the durability coefficient of pheromone; Δt_{ij} represents the total increment of pheromone on the path between city i and city j in this round of iteration; Δt_{ij}^{k} denotes the pheromone amount left by ant k on the path between city i and city j in this round of iteration(if ant k didn't travel along path(i, j), then Δt_{ij}^{k} is 0). The detailed steps of ant colony optimization algorithm for solving the TSPs are as follows:

(1)Initialize parameters. Let t=0(clock time), $ncout$=0(counter of iteration), and make assignment to maximum times of iteration $ncout_{max.}$ Place m ants randomly at n cities. Let initial amount of pheromone on each path $t_{ij}(t)$=const, where const is a constant, and $\Delta t_{ij}(0)$=0 ; .

(2)Make assignment to counter of iteration: $ncout \leftarrow ncout$ +l ;

(3)$index$=1 ;

(4) Make assignment to the index of the ant in play: $k \leftarrow k$+1;

(5) Ant k makes its decision of which neighboring city to visit next, say city $j(j \in \{C\text{-}tabu_k\})$, according to probabilities shaped by Formula ①;

(6) Move ant k to city j and add city j into $tabu_k$, tabu list for ant k;

(7)If $k<m$, which means not all cities are visited in this round of visit, go to step (4). Otherwise go to step (8).

(8)Update pheromone on each visited path according to Formula ② and Formula ③.

(9)If stopping rule, such as $ncout \geq ncout_{max}$, has been reached, then stop and output the path picked by the majority of ants. Otherwise empty tabu lists go to step (2).

Conclusions

Natural computation algorithms are different from traditional ones in many aspects, intelligence and essential parallelism in particular. Intelligence includes self-organization, self-adaptation, self-learning and etc. The essential parallelism lies in two aspects: One is inherent parallelism, that is, the nature of natural computation itself fits the requirement of massive parallel processing; The other is implicit parallelisms, which means that natural computation algorithms search for the best solution in a population using a iterative progress guided by information exchange. So its algorithms can process searching in approximate $O(N^3)$ times with computing operations proportional to N(the population size), which makes them good solvers for intelligent optimization problems such as machine learning, adaptive control and etc. The application of natural computation for solving the TSPs is of significant value for related research on how to construct the framework of evolutionary algorithms and to introduce effective evolution operations.

Acknowledgements

This work was supported by the the National Natural Science Foundation of China under grant No. 71561013;the Natural Science Foundation of Jiangxi
Province under grant No. 20151BAB207040; and the Subject of Teaching Reform in Universities of Jiangxi Provincial Education Department under grant No. JXJG-14-10-9.

References

[1] C.Z. Fan.The research of evolution algorithm for TSPs. A dissertaion submitted to Hunan normal university for the degree of master,2007.

[2] K.Q.Liu,L.S.Kang,Z.Z.Zhou.The research brief (I)On the branches of cognitive evolution computing.computer science, Vol.36(7):26-31,July,2009.

[3] S.Q.Liu,K.Y.Yang.the Research on Improved Genetic Algorithm for solving TSPs.Journal of Beijing university of information technology(natural science edition),Vol.29(2):46-50,Apr,2014.

[4] A.Gong,M.Zhang. A TSP method for Hopfield network based on Clustering Technology. Computer simulation, Vol.23(08):174-176,Aug,2006.

[5] Z.P.Yang,Y.L.Huang,L.G.Qu,P.P.Ge. The research of improved ant colony algorithm for TSP and its simulation. Vol.37(08):928-932,Aug,2014.