An ant inspired algorithm for solving Suggestion-sorting Problem

Sichao Song

1. Introduction

For the suggestion-sorting problem, the difficulty is how to come up with a user-satisfied sequence of suggestions in real-time premise considering that the problem set is continuously augmenting. In this article I will introduce ant colony optimization and survey its advantages on this type of problems over its counterparts.

1.1 Ant inspired optimization (ACO)

ACO is the algorithm inspired by the foraging behavior of ants. An ant in its colony can find the shortest path between a food source and their nest by following the "stimuli", so called pheromone, which is laid by the other ants in the same swarm. In ACO, a number of artificial ants build solutions to an optimization problem and exchange information on their quality via a communication scheme, which is called artificial pheromone, in order to find a near-to-best solution in a very short time. Considering the suggestion-sorting problem which undoubtedly requires real-time interaction with the mobile user, ACO is a good choice.

1.2 Markov assumption

Assume that the probability of an incident only depends on the immediately preceeding incidents within some window of length n:

$$P\left(w_k|w_1^{k-1}\right) \approx P\left(w_k|w_{k-n+1}^{k-1}\right)$$

For example, we have 1 sequence including 5 suggestions S=A->B->C->D->E, when n=2, we have:

$$P(s)=P(A)*P(B|A)*P(C|A,B)*P(D|A,B,C)*P(E|A,B,C,D)$$
$$\approx P(A)*P(B|A)*P(C|B)*P(D|C)*P(E|D)$$

Then the probability of P(s) is easy to compute (we can design an experiment for collecting data for observing those probabilities such as P(B|A), P(C|B) etc.)

1.3 Why we need to sort suggestions?

When the user is involved in a scenario, our system (e.g. mobile phone) needs to come up with a sequence of suggestions for the user to choose. Sometimes the sequences could be so long that we only want to display a small part of them in the screen. Thus, this sequence of suggestions should be ordered by probabilities, or in

other words, to fit the user's preference. Typically, when in frequent scenarios, the systems need to give the user same suggestions repeatedly, then it will be a much better idea to sort the suggestions based on user's preference in order to humanize our systems.

2. The Ant colony optimization algorithm (ACO) modeling

2.1 Probability modeling by Bayers' rule

The basic idea is sorting suggestions based on probability. Assume that we have a set of suggestions $\{m_i | m_{i\in[1,5]}=A, B, C, D, E\}$ and the sequence of suggestions $\{s | s=m_1->m_2->m_3->m_4->m_5\}$. Because that each suggestion can be selected repeatedly by the user, we have totally 5^5 (=3125) different combinations of suggestions. Assume that for each time t, the system gives a new sequence of suggestions for the user, we have the probability of s(t):
When at time t,

$$P(s(t))=P(m_1)*P(m_2|m_1)*P(m_3|m_1,m_2)*P(m_4|m_1,m_2,m_3)*P(m_5|m_4)$$
$$\approx P(m_1)*P(m_2|m_1)*P(m_3|m_2)*P(m_4|m_3)*P(m_5|m_4)$$

Now we have the probability of the current sequence, we want to know the probability of the next suggestion. By using Bayers' rule, we have:

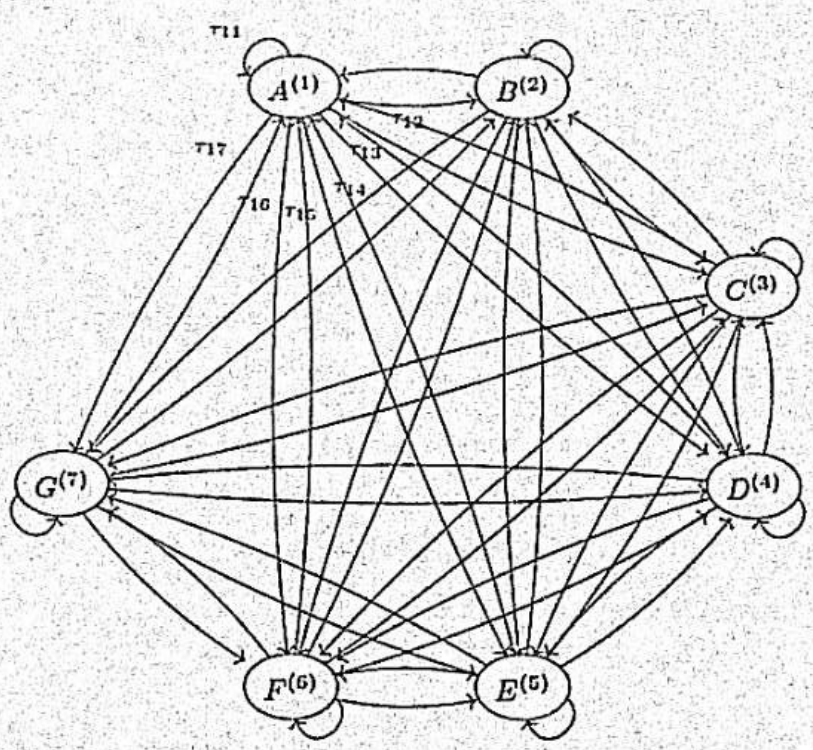
$$P(s(t+1)|s(t))=\frac{P(s(t)|s(t+1))*P(s(t+1))}{P(s(t))}$$

We already have P(s(t)) and P(s(t+1)) (in this case is 1/5, or can be observed from experiment data). Here the key problem is to compute the "likelihood" P(s(t)|s(t+1)). Now the problem looks somehow like a classification problem, that is giving the previous sequence of suggestions, which class (in this case m_1, m_2, m_3, m_4, m_5) should we assign the next suggestion to? One solution that comes to my mind is also establishing a training set for it. But it is hard to apply in real life because the training set could probably be so huge and such an classification could take quite a long time. So for modeling this "likelihood" P(s(t)|s(t+1)), artificial ants are now coming into play.

2.2 Artificial ants on the graph of suggestions

We use artificial ants to build a sequence of suggestions according to transition probabilities and taking advantage of the collective behavior of making paths with pheromones. Artificial ants are agents that are located on vertices in a graph and can move and deposit pheromones on those edges. In this case, we only use one ant because each time we only need to generate one new suggestion (we can also use more than one ant and put these additional ants as "ghosts" that not generate

suggestions but only searching in the graph). The vertices correspond to suggestions (A, B, C, D, E) and each edge (i,j) between vertices i and j, is weighted by a positive pheromone value τ_{ij} (see figure below).



A transition rule is used to decide which of the vertices an ant will choose when located in vertex i, the probability of choosing vertex j is given by:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{k \in N_i} \tau_{ik}^{\alpha} * \eta_{ik}^{\beta}}$$

where η_{ij} is a numerical value representing the desirability of the ant choosing the edge from i to j. The parameter α, β controls the influence of pheromone and the desirability: high/low values increase/decrease the importance of its parameter. Finally, N_i stands for the set of possible vertices that can be reached from i.

Each time an ant moves from a vertex i to a vertex j, it deposits a pheromone quantity τ_0:

$\tau_{ij} \leftarrow \tau_{ij} + \tau_0$

Finally, as it occurs in natural systems, pheromone slowly evaporates:

$\tau_{ij} \leftarrow (1 - \rho) \times \tau_{ij}$

where ρ is a parameter called evaporation ($0 \leq \rho \leq 1$).

Now if you remember the "likelihood" P(s(t)|s(t+1)), you can see that we are building

the training set and training the "classification" while searching in the graph! The priori knowledge is contained in our ant model thus the posterior probability p_{ij} is then easily computed.

Here is one thing very import that while artificial ant depositing pheromone τ_0 during its searching, user's pheromone τ_u should also be deposited on the edge between his/her last selection and the current selection. This τ_u should much more greater than τ_0 because our purpose is to let the user "lead" the ant in order to generate the sequence of suggestions according to the user's preference. Thus, the pheromone update should be:

$\tau_{ij} \leftarrow \tau_{ij} + \tau_0 + \tau_u (\tau_u >> \tau_0)$

Each time when the system about to give a new sequence of suggestions, the artificial ant searches for one run and choose a new suggestion (the ant may go to another vertex or stay in the same vertex). At time t, we have a sequence of length t, and at time t+1 the length increases to t+1. The new sequence has its first element of the new generated suggestion, and the rest t elements are the same as the last sequence at time t. However, when the sequence is displayed in the terminal monitor, the repeat elements should be eliminated. For example, we have a sequence A->B->B->D->B->B, if the screen can only display 2 suggestions, then it should be {B, D}.

3. Conclusion

Ant colony optimization algorithm (ACO) is quite suitable for typically routing problems. While there are some other possible ideas come into my mind e.g. K-means classification, I am convinced that ACO is more applicable and has advantages over its counterparts. It is very effective for NP-hard problems and also comfortable with continuous optimization problems such as our suggestion-sorting problem. In this article I only applied one artificial ant for searching, but we can also try using more ants (one to generate suggestion and the others are collaborating for exploring the graph).

One drawback of this system is that it is essentially a probability model, which is inevitable that the solutions that have low probabilities also can be selected. One possible way is to set a threshold in order to let the system gives solutions only have higher probabilities.