Development of an automatically configurable ant colony optimization framework. State of art.

Aldar Saranov

March 15, 2017

Contents

1 Introduction (1 page) 3

2 Combinatorial Optimization Problems and Constructive Heuristics (17 pages) 3

3 The Concepts of Ant Colony Optimization 6
3.1 Choice of pheromone trails and heuristic information 6
3.2 Solution construction 6
3.3 Global pheromone update 7
3.4 Pheromone update schedule 8
3.5 Initialization of pheromones 8
3.6 Pheromone reinitialization 8
3.7 Local pheromone update 8
3.8 Pheromone limits 8
3.9 Local search 8
3.10 ACO algorithms as instantiations of the ACO Metaheuristc . . 9

4 Applications of ACO to other problem types 9
4.1 Continuous Optimization Problems 9
4.2 Multi-objective problems 9
4.3 Dynamic problems 9
4.4 Stochastic problems 9

5 ACO in combination with other methods 9
5.1 ACO and tree search methods 9
5.2 ACO and exact methods 9
5.3 ACO and surrogate models 9
5.4 Parameter adaptation 9

6 Existing ACO framework (5 pages) 9
6.1 Finding a better ACO configuration for the TSP 9
6.2 Finding a better ACO configuration for the QAP 9

7 IRACE automatic configuration (3 pages) 9

8 Conclusions 9

Abstract

Some animal species show an extreme degree of social organization. Such species (e.g. ants) have pheromone production and detection body parts and therefore seize an ability to communicate between each other in indirect way. This concept has inspired the development of algorithms which are based on social behavior of population called ant colony optimization algorithms (ACO). These algorithms allow to solve NP-hard problems in a very efficient manner. Since these algorithms are considered metaheuristic the development of a ACO framework is the next step of formalizing of this area is to provide tools for resolving general optimization problems. This article gives the brief overview of the current ACO research area state, existing framework description and some tools which can be used for the framework automatic configuration.

1 Introduction (1 page)

Section descriptions. Pheromones. Constructive heuristics. Solution components. Problem models.

2 Combinatorial Optimization Problems and Constructive Heuristics (17 pages)

Combinatorial optimization problems (COP) are a whole class of mathematical optimization problems. These problems can be described by grouping, ordering, assigning or any other operations over the set of discrete objects. In practice one may need to resolve COP which have a large number of extra constraints for the solutions which are considered admissible. Many of these problems which are being thoroughly researched at the moment belong to NP-complete discrete optimization problems. NP-complete problem informally means that we cannot decompose a large instance of such problems into a smaller one. Algorithms solving such problems take time larger than polynomial (e.g. exponential).

Definition

Optimization Problem is a tuple (Φ, ω, f), where

- Φ is a search space consisting of all possible assignments of discrete variables x_i, with $i = 1, ..., n$
- ω is a set of constraints for the decision variables
- $f : \Phi \to R$ is an objective function which has to be optimized

The problem describes the abstract subclass of tasks (e.g. find the minimum spanning tree of some graph) while the instance of a problem describes a certain practical problem (e.g. find the minimum spanning tree of a given graph G). The objective function in this case is the sum of the selected edges. One of the most frequently encountered problems is traveling salesman problem (TSP). Given a graph $G = (N, E)$ with $n = |N|$ nodes, where E - is a set of edges fully connecting the nodes and distances $d_{ij}, \forall(i, j) \in E$ one should find a Hamiltonian path of minimal length (in terms of sum of the weighted edges). The solution path can be represented as $\pi = (\pi_1, ..., \pi_n)^t$ of all n nodes, where pi_i is the node index at position i. The optimal value of the objective function is

$$\min_{\pi \in \Phi} d_{\pi_i \pi_{i+1}} + \sum_{i+1}^{n-1} d_{\pi_i \pi_{i+1}} \tag{1}$$

Thus π forms a permutation space and every permutation of π gives a admissible (but not necessarily optimal) solution. Plus it is obvious that the absolute position in the permutation sequence does not affect the value of the objective function but the relative one.

In addition to TSP the quadratic assignment problem (QAP) was deeply researched. In QAP there is a set of n locations and a set of n facilities which are connected with $n \times n$ flows. The objective function is represented as the sum of paired production of distances between i and j locations and specified flows between π_i and π_j assigned flows. An instance of the problem is given $n \times n$ matrices d_{ij} and f_{ij} with $i, j = 1, ..., n$. A solution of QAP is assignment of the facilities to the locations represented by permutation π where π_i depicts assignment of the corresponding facility to the location i.

$$\min_{\pi \in \Phi} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \times f_{\pi_i \pi_j} \tag{2}$$

Solution components are normally defined in the terms of COP. Solution components $C = c_1, c_2, ...$ is a set, subset of which corresponds to one so-

lution of the given problem (if also fulfills the constrains). Solutions that fulfill all the constraints are also called *feasible solutions*. In case of TSP solution components are the edges (i, j) of the given graph. In case of QAP solution components are all the possible assignments of every location i to every facility j. In order to provide the feasible solutions the algorithm must either to operate completely in the feasible candidate solution space or to bias towards the feasible ones with final constraint checking.

Since solving of such problems by using provably optimal solutions is unreasonable one can apply heuristic algorithms which more or less provide solutions with relatively good fitness consuming reasonable quantity of resources (time/power, memory etc.).

An essential way in such cases is using of constructive heuristics. Constructive heuristics starts with an empty or partially built solution and is being completed by iterated extension until finished. Each of the iterations adds one or several solution components to the solution. For example greedy constructive heuristics algorithm adds best-ranked component and therefore provides high level of exploiting.

In application to the TSP problem *nearest neighbor* heuristic is used. The algorithm starts from random node i with initial $pi =<>$, at each step it selects the solution component with the minimal distance d_{ij} and adds the corresponding $\pi_1 = i$ and $\pi_2 = j$ components to the solution.

In application to QAP one tends to place the facilities at the locations that are more "central". The algorithm computes $f = (f_1, ..., f_n)^t$ where $f_i = \sum_{j=1}^{n} f_{ij}$ and $d = (d_1, ..., d_n)^t$ where $d_k = \sum_{l=1}^{n} d_{kl}$. Later on the algorithm assigns the facilities with the largest f_i to the locations with smallest d_k.

Generally heuristic values are assigned constants, however in extensions one can use heuristics which is a function of the generated partial solution as input. This is called *adaptive* heuristics and normally it consumes larger computer resources although leads to better quality of the solutions.

3 The Concepts of Ant Colony Optimization

Algorithm

procedure ACO-Metaheuristic
    repeat
        for each ant **do**
            repeat
                ExtendPartialSolutionProbabilistically()
            until solution is complete
        for each ant ∈ SelectAntsForLocalSearch() **do**
            ApplyLocalSearch(ant)
        EvaporatePheromones()
        DepositPheromones()
    until termination criteria met
end

3.1 Choice of pheromone trails and heuristic information

C - Solution components.
$\tau_c \in T$ - pheromones of choosing.
$\tau'_c \in T'$ - pheromones of considering order.
π - candidate solution.
$\eta_c \in H$ - heuristic information (constant in time).

3.2 Solution construction

A solution is constructed by an ant.
Probabilistic rules:

- Classic
- Maniezzo
- Dorigo

α, β - choice parameters.
Extensions:

- Lookahead - pick several components at once[94]
- Candidate list - restriction of component choice at each step[33,34]
- Iterated greedy (partial deconstruction)[110]
- With external memory[1]
- Iterated ants[129]
- Cunning ants[128]
- Enhanced ACO[47]

3.3 Global pheromone update

Evaporation:
$\tau_{new} = evaporation(\tau_{old}, \rho, S^{eva})$
ρ - evaporation rate
S^{eva} - chosen solutions for evaporation

Deposition:
w_k - weight of solution s_k.
$F(S_k)$ - non-decreasing solution quality scaling function.

Update selection:

1. Ant system (update all)
2. Single update selections:
   (a) iteration-based update
   (b) global-based update
   (c) restart-based update

Update extensions:

1. Max-Min Ant System [122]
2. Rank-based Ant System [19]
3. Best-Worst Ant System [21]
4. Elitist Ant System [30, 36, 38]

3.4 Pheromone update schedule

Exploration vs exploitation.

3.5 Initialization of pheromones

3.6 Pheromone reinitialization

3.7 Local pheromone update

Parallel vs sequential. ACS [34]

3.8 Pheromone limits

MMAS and ACS examples.

3.9 Local search

Neighborhood operator. Best-improving and first-improving.

3.10 ACO algorithms as instantiations of the ACO Meta-heuristc

4 Applications of ACO to other problem types

4.1 Continuous Optimization Problems

4.2 Multi-objective problems

4.3 Dynamic problems

4.4 Stochastic problems

5 ACO in combination with other methods

5.1 ACO and tree search methods

5.2 ACO and exact methods

5.3 ACO and surrogate models

5.4 Parameter adaptation

6 Existing ACO framework (5 pages)

6.1 Finding a better ACO configuration for the TSP

6.2 Finding a better ACO configuration for the QAP

7 IRACE automatic configuration (3 pages)

8 Conclusions

References