Method for Optimizing Communication Costs in ACODA Using Simulated Annealing

Costin Bădică
University of Craiova
Software Engineering Department
Bvd. Decebal 107, Craiova
200440, Romania
cbadica@software.ucv.ro

Sorin Ilie
University of Craiova
Business Informatics and Statistics Department
A.I.Cuza 13, Craiova
200585, Romania
silie@software.ucv.ro

Mirjana Ivanović
University of Novi Sad
Faculty of Sciences
Department of Mathematics and Informatics
Trg Dositeja Obradovica 3
21000 Novi Sad, Serbia
mira@dmi.uns.ac.rs

ABSTRACT

ACODA is a truly distributed framework for Ant Colony Optimization. ACODA is heavily using message passing, so communication costs are quite high. In this paper we formulate the optimization of communication costs in ACODA as a mathematical optimization problem. We analyze the feasibility of its solution using Simulated Annealing.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

Keywords

Algorithms, artificial intelligence, distributed systems

1. INTRODUCTION

Ant Colony Optimization on a Distributed Architecture – known as ACODA is a multi-agent distributed framework for Ant Colony Optimization (ACO hereafter) [4]. ACODA was developed for solving computationally difficult path optimization problems in graphs, such as the Traveling Salesman Problem (TSP hereafter) [6]. The main idea behind ACODA was to represent the network of nodes that define the problem domain as a distributed multi-agent system. In particular, the agents are able to store and exchange ant information, thus enabling a fully distributed model of ants migration as message exchanged by agents.

In fact, with our proposal we identify and exploit fine-grained parallelism that is inherently present in collective natural bio-systems, in particular in colonies of natural ants that is the metaphor behind ACO. According to this view, ACODA can be conceptualized as a network of computational nodes that are exchanging ant information according to ACO rules. So, using parallel computing as source of inspiration, we observed that by defining a suitable partitioning of the nodes of this network and its mapping to a high-speed computer network, we can efficiently divide the communication and computation costs of ACODA among the available network machines, thus leading to the improvement of the overall execution time of an ACO algorithm. The problem of defining a suitable partition of the network of ACODA computational nodes is an instance of the Graph Partitioning problem (GP hereafter) – a well-known NP-hard computational problem [3].

In this paper we propose the use of Simulated Annealing (SA hereafter) for defining a suitable partition of ACODA computational nodes. SA is a probabilistic mathematical optimization method inspired by the physical process of slowly cooling a metal until its internal structure is frozen to an equilibrium state that represents the optimal solution of the problem [2]. SA has already been applied to solve the GP problem [5].

Note however that there are various approaches for defining the optimization criteria of GP [3]. In this paper we define an optimization criteria suitable for ACODA, as well as a new formulation of the solution space of SA. Then we propose an SA algorithm inspired by [5] for exploring the partitioning solution space. We also present initial experiments of applying this algorithm on real data extracted during the execution of ACODA while solving standard benchmark instances of the TSP problem [6].

It is interesting to note that there is an important difference between the standard application of GP to parallel computing problems and its application to optimization of communication costs in ACODA. The standard application assumes that the optimization graph and the partitioning are defined and computed *before* the execution of the parallel problem solving process. On the other hand, ACODA can be described as a probabilistic search algorithm, so it is impossible to define the graph and the partitioning before the algorithm is run. With our approach the partitioning should be done *during* rather than *before* the execution of the parallel problem solving process.

2. BACKGROUND

Following [4], ACODA architecture can be succinctly described as a network of computational nodes mapped to several physical machines interconnected by a high-speed communication network. The structure of the network of computational nodes mimics the graph that defines the instance of the optimization problem (TSP in this case) which ACODA is configured to solve. Basically, pairs of nodes can exchange messages (i.e. ant information). The set of nodes is distributed to the set of the available network machines. Usually, the set of nodes is much larger than the set of available machines. In our JADE-based implementation [1] of ACODA each machine is running a JADE agent that is managing the set of nodes assigned to that machine.

The cost of exchanging messages between two nodes located on different physical machines is obviously significantly higher than the cost of exchanging messages between two nodes located on the

BCI'12, September 16–20, 2012, Novi Sad, Serbia.

Local Proceedings also appeared in ISBN 978-86-7031-200-5, Faculty of Sciences, University of Novi Sad.

same physical machine. Intuitively, if two nodes exchange a large number of messages they should be located on the same machines, while if they exchange a reduced number of messages they can be safely located on different machines. Therefore the optimal partitioning of the set of nodes should take into account the specific interaction pattern between the nodes that is defined by the execution of ACODA.

3. OPTIMIZATION OF COMMUNICATION COSTS

We model ACODA as a network of n computational nodes that are mapped onto a high-speed computer network consisting of k physical machines (or processors). The mapping of nodes to machines can be described by a partition p of the set $\{1, 2, \ldots, n\}$ into k sets. For each $i \in \{1, 2, \ldots, k\}$ let p_i be the set of nodes mapped onto machine i.

During the execution of ACODA we record the number of messages (or ants) m_{ij} exchanged by any two nodes i and j. m is an $n \times n$ symmetric matrix. The nodes of the set p_i that are mapped to machine i will exchange messages with the rest of nodes, i.e. $\bar{p_i} = \{1, 2, \ldots, n\} \setminus p_i$, so the associated cost can be computed as:

$$C_i(p) = \sum_{j \in p_i, l \in \bar{p_i}} m_{jl}$$

The total communication cost incurred during the execution of ACODA can be estimated as the total number of messages exchanged by the machines, as follows:

$$C(p) = \sum_{i=1}^{k} C_i(p)$$

Let $0 < d < n$ be a natural number and let $q = \lfloor n/k \rfloor$, i.e. q is the quotient of the integer division of n by k. A partition p is called d-balanced if for each $i \in \{1, 2, \ldots, k\}$ the number of elements of the set p_i is bounded as follows: $||p_i| - q| \leq d$. This means that the number of elements of p_i cannot be larger than $q + d$ and it cannot be smaller than $q - d$. For example, if $d = 1$ then each set of a 1-balanced partition can contain either $q - 1$, q or $q + 1$ elements.

The GP problem asks for computing a d-balanced partition p such that the total communication cost $C(p)$ is minimum.

4. SIMULATED ANNEALING

SA is a probabilistic mathematical optimization method inspired by the physical process of slowly cooling a metal until its internal structure is frozen to an equilibrium state that represents the optimal solution of the problem [2]. We follow the formulation of the SA algorithm presented in [5].

The key ingredients of the SA algorithm are as follows:

- The set S of feasible solutions also known as "problem states". In our case this is given by the set p of d-balanced partitions.
- For each $x \in S$ the set $S(x)$ of its "neighbors". In our case a neighbor is defined by choosing two distinct sets p_i and p_j of the partition such that $|p_i| > q - d$ and $|p_j| < q + d$ and moving one arbitrary element from p_i to p_j. It is easy to see that the neighbor is correctly defined, i.e. it is a d-balanced partition. Moreover we can always choose p_i as the set with the largest number of elements and p_j as the set with the smallest number of elements that shows that in our formulation of the GP problem each state has at least one neighbor.
- A nondecreasing "temperature" function $T : \mathbb{N} \to \mathbb{R}^+$ also known as "cooling schedule".

SA starts from a given initial state. It proceeds through an iterative search process without backtracking by randomly choosing in each iteration a neighbor of the current state. SA seeks to improve (i.e. to decrease the value of the cost of) the currently best solution. So most of the time SA will prefer downhill moves. However, in order to avoid being trapped to local minima, sometimes SA allows uphill moves. The selection of uphill moves is controlled by the cooling schedule. SA stops when the improvement of the currently best solution seems highly unlikely (the standard term is that the state is "frozen").

The SA algorithm is illustrated in Fig. 1. The main source of inspiration for this algorithm is [5]. It takes an initial state p and returns an optimized state p_{min}.

The SA algorithm is using several functions as follows:

- *neigh_size*() that determines an estimation of the number of neighbors of a problem state;
- *successor* that randomly picks up a neighbor $q \in S(p)$ of p
- *cost* that determines the value of the current state
- *random* that determines a uniformly distributed random number.

The SA algorithm can also be externally configured using the following three parameters:

- *SIZEFACTOR* is a real number between 0 and 1 that represents the proportion of neighboring states from the estimated number of neighbors that will be handled in the current iteration
- *MAX_CNT* that represents the maximum number of times the algorithm tries to improve the currently best solution during the current iteration before declaring that the state is frozen
- *MIN_PERCENT* that determines the minimum proportion of moves that must be accepted during the current iteration in order to consider that the further improvement of the currently best solution is still possible, i.e. the state is not frozen.

A crucial aspect of any SA algorithm is the cooling schedule. There are several proposals in the literature:

- *Linear* cooling schedule, defined as:

$$T_{lin}(i) = T_0/i$$

- *Logarithmic or Boltzman* cooling schedule, defined as:

$$T_{boltz}(i) = T_0/(1 + \log i)$$

- *Exponential* cooling schedule, defined as:

$$T_{exp}(i) = T_0 * \gamma^i, \;\; 0 < \gamma < 1$$

```
SIMULATED-ANNEALING(p)
1. C ← cost(p), C_min ← C, p_min ← p, cnt ← 0
2. i ← 1, trials ← SIZEFACTOR * neigh_size()
3. while cnt < MAX_CNT do
4.       T ← temp(i)
5.       acc_moves ← 0
6.       for i = 1, trials do
7.           q ← succesor(p), C_1 ← cost(q), Δ ← C_1 − C
8.           if Δ < 0 then
9.              p ← q, C ← C_1, acc_moves ← acc_moves + 1
10.             if C < C_min then
11.                p_min ← p, C_min ← C, cnt ← 0
12.          else
13.             prob ← exp(−Δ/T)
14.             r ← random()
15.             if r ≤ prob then
16.                C ← C_1, p ← q
17.                acc_moves ← acc_moves + 1
18.      if acc_moves ≤ trials * MIN_PERCENT then
19.         cnt ← cnt + 1
20.      i ← i + 1
21. return p_min
```

Figure 1: SA algorithm.

Before usage, the cooling schedules must be calibrated by determining a suitable value for the initial temperature T_0 (parameter *gamma* is set in advance to a value smaller than and close to 1, for example 0.95). The calibration process can be achieved by choosing an initial state p and estimating the difference between $C(p)$ and the average value of the neighboring states q of p, i.e.:

$$\Delta = (\sum_{q \in S(p)} C(q))/|S(p)| - C(p)$$

Then, given an initial value for the probability r of accepting uphill moves (used in steps 13-17 of SA algorithm from Fig. 1; usually this is taken as 0.2), Δ can be used to compute T_0 by solving the following equation:

$$r = exp(-\Delta/T(1))$$

for each of the cooling schedules: T_{lin}, T_{boltz} and T_{exp}.

Finally, we can provide an estimation of the size of the neighborhood of a given partition p (the value returned by the function *neigh_size*()) for the GP problem. A neighbor can be in principle determined by choosing an element x of one of the sets of the partition and transferring it to one of the other $k-1$ sets of the partition. Taking into account that there are n elements then we can estimate the size of the neighborhood for the GP problem as $n * (k-1)$.

5. CONCLUSION AND FUTURE WORK

We have presented our approach and work-in-progress for optimizing communication costs in ACODA – a multi-agent distributed framework for Ant Colony Optimization. We are now in the process of performing an experimental evaluation of this approach. We shall report on our progress in subsequent papers.

6. ACKNOWLEDGMENTS

This work was partially supported by the multilateral agreement on academic cooperation in 2012 between Serbia (Novi Sad), Romania (Craiova), and Poland (Warsaw and Gdansk) on "Agent Technologies, Tools, Environments, Applications".

7. REFERENCES

[1] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons Ltd, 2007.

[2] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993.

[3] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519–1534, 2000.

[4] S. Ilie and C. Bădică. Multi-agent approach to distributed ant colony optimization (in press). *Science of Computer Programming*, 2011.

[5] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation. part i, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[6] G. Reinelt. Tsplib - a traveling salesman library. *ORSA Journal on Computing*, 3:376 – 384, 1991.