Computers & Operations Research 40 (2013) 931–942

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Countering the negative search bias of ant colony optimization in subset selection problems

Jan Verwaeren *, Karolien Scheerlinck, Bernard De Baets

KERMIT, Department of Mathematical Modelling, Statistics and Bioinformatics, Ghent University, Coupure Links 653, Ghent, Belgium

ARTICLE INFO

Available online 6 November 2012

Keywords:
Evolutionary computations
Combinatorial optimization
Ant colony optimization
Search bias
Subset problems

ABSTRACT

In their quest to find a good solution to a given optimization problem, metaheuristic search algorithms intend to explore the search space in a useful and efficient manner. Starting from an initial state or solution(s), they are supposed to evolve towards high-quality solutions. For some types of genetic algorithms (GAs), it has been shown that the population of chromosomes can converge to very bad solutions, even for trivial problems. These so-called deceptive effects have been studied intensively in the field of GAs and several solutions to these problems have been proposed. Recently, similar problems have been noticed for ant colony optimization (ACO) as well. As for GAs, ACO's search can get biased towards low-quality regions in the search space, probably resulting in bad solutions. Some methods have been proposed to investigate the presence and strength of this negative bias in ACO. We present a framework that is capable of eliminating the negative bias in subset selection problems. The basic Ant System algorithm is modified to make it more robust to the presence of negative bias. A profound simulation study indicates that the modified Ant System outperforms the original version in problems that are susceptible to bias. Additionally, the proposed methodology is incorporated in the Max–Min AS and applied to a real-life subset selection problem.

1. Introduction

Most population-based metaheuristics will initially search for solutions in a random manner. Because of this, the agents in the first generations will often find solutions of intermediate quality. However, the results of this initial random search will influence the agents in later generations. As a result, the agents of these later generations are biased; they will concentrate their search on some part of the search space. Since metaheuristics are intended to find high-quality solutions, it is desirable that the focus of the search is directed towards the promising regions in the search space. However, it has been noticed that evolutionary algorithms can evolve towards less promising regions. For genetic algorithms (GAs) this phenomenon was called *deception* in [1]. In [2], the concept of deception is generalized to ant colony optimization (ACO).

Up to now, research on the avoidance of bias in ACO is limited. It has been argued on multiple occasions [2–4] that the choice of a suitable pheromone representation can mitigate the effects of negative search bias. However, as far as we know, there is no evidence that the choice of a suited pheromone representation on its own can eliminate all the deceptive effects. Therefore, we introduce a new kind of deception avoidance strategy that is suited for solving subset selection problems, such as for instance the well-known knapsack problem [5], with ACO. However, our bias-avoidance mechanisms are applicable to a wide range of problems as illustrated in the experimental section.

This paper is organized as follows. In Section 2, we give a brief review of the Ant System algorithm. In the third section, bias in metaheuristics is considered in general and discussed more in detail in the light of ACO. In Section 4, modifications of the basic Ant System algorithm are proposed to counter negative search bias. Sections 5 and 6 describe the setup and results of a wide range of experiments. Finally, some conclusions are drawn in Section 7.

2. Ant System

Ant System (AS) was the first real ACO algorithm, inspired on the behaviour of real ants, and was introduced by [6]. Since its development, this basic version of ACO has been altered numerous times in order to increase its performance or to adapt it to specific problem settings [7]. Although it is sometimes argued that the performance of AS is inferior to younger variants such as the Max–Min Ant System (MMAS) [8], we will mainly use it as the basic algorithm for this study. This choice seems justified since AS contains all the basic ACO elements. Moreover, even in recent

* Corresponding author. Tel.: +32 92645931; fax: +32 92646220.
E-mail addresses: Jan.Verwaeren@UGent.be (J. Verwaeren), Karolien.Scheerlinck@UGent.be (K. Scheerlinck), Bernard.DeBaets@UGent.be (B. De Baets).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.10.019

studies [9,10], AS is still used and achieves state-of-the-art results. Additionally, it is the simplest form of an ACO algorithm. Moreover, rather than constructing a new state-of-the-art algorithm, our main goal is to study the behaviour of ACO algorithms in general. Nevertheless, we can incorporate our methodology into ACO variants such as MMAS, as will be illustrated in the application section.

2.1. Basics and notations

Let us start with the formal definition of a subset selection problem. A subset selection problem described by a triple (S,f,g), with item set S, objective function $f: 2^S \to \mathbb{R}$ and constraint function $g: 2^S \to \{0,1\}$ consists of the following optimization problem:

$$\max_{\mathbf{x} \in 2^S} f(\mathbf{x}) \quad \text{s.t. } g(\mathbf{x}) = 1.$$

In the following sections, subsets of S will often be denoted as n-tuples, requiring an ordering/indexing of the elements of S, i.e. $S = \{s_1, \ldots, s_n\}$. Formally, a subset $\mathbf{x} \in 2^S$ can be written as an n-tuple $\langle s_1^{j_1}, \ldots, s_n^{j_n} \rangle$, where $j_i = 1$ if $s_i \in \mathbf{x}$ and $j_i = 0$ if $s_i \notin \mathbf{x}$. Let $\mathbf{x}_i$ be the i-th component of this n-tuple. We have that $\mathbf{x}_i \in X_i = \{s_i^0, s_i^1\}$. Denoting $\mathcal{X}_i = X_1 \times \cdots \times X_i \quad (i \leq n)$ we can write $2^S \equiv \mathcal{X}_n = X_1 \times \cdots \times X_n$. Using this tuple notation, we define the feasible space $\mathcal{X}_n^f = \{\mathbf{x} \in \mathcal{X}_n | g(\mathbf{x}) = 1\}$. Now consider the set $S_i = S \backslash \{s_{i+1}, \ldots, s_n\}$ and $\mathbf{y}^i \in 2^{S_i}$. $\mathbf{y}^i$ can be written as an i-tuple: $\mathbf{y}^i = \langle s_1^{j_1}, \ldots, s_i^{j_i} \rangle \in \mathcal{X}_i$. Moreover, let $\mathbf{y}_{k]}^i = \langle s_1^{j_1}, \ldots, s_k^{j_k} \rangle$ be the first k $(k \leq i)$ components of the i-tuple $\mathbf{y}^i$, then $\mathbf{y}_{k]}^i \in \mathcal{X}_k$. For i-tuples, the feasible space is defined as $\mathcal{X}_i^f = \{\mathbf{y}^i \in \mathcal{X}_i | (\exists \mathbf{x} \in \mathcal{X}_n^f)(\mathbf{x}_{i]} = \mathbf{y}^i)\}$. Finally, using the concatenation operator $\oplus$ on tuples, we can write $\mathbf{x} = \mathbf{x}_{i]} \oplus \langle s_{i+1}^{j_{i+1}}, \ldots, s_n^{j_n} \rangle$.

2.2. AS for subset selection

In this section we will briefly describe (our interpretation of) AS for subset selection, closely following the original AS proposed in [6]. Pheromone trail parameters are a key ingredient of AS. In this paper, we assign a pheromone trail parameter τ_i^j to each s_i^j where $\tau_i^j(t) \in \mathbb{R}^+$ denotes the value of this parameter at generation (iteration) t. Moreover, we will use T to refer to the complete matrix of pheromone trail parameters (and $T(t)$ their values at time t). Since ACO is a constructive metaheuristic, each individual agent (ant) builds its own solution starting from scratch. As such, an agent starts with an empty partial solution (a tuple of length zero) $\mathbf{y}^0 = \langle\,\rangle$ and extends it through concatenation at each construction step. During the i-th $(i = 1, \ldots, n)$ construction step, the tuple $\mathbf{y}^i \in \mathcal{X}_i$ is constructed as $\mathbf{y}^i = \mathbf{y}^{i-1} \oplus \langle s_i^j \rangle$ (with $j \in \{0,1\}$ and $\mathbf{y}^{i-1} \in \mathcal{X}_{i-1}^f$), where s_i^j is the result of a probabilistic decision rule, parameterized by T:

$$P(s_i^j | \mathbf{y}^{i-1}, T(t)) = \frac{\tau_i^j(t)\, \eta(s_i^j)}{\sum_{\{s_i^k \in \mathcal{E}_{\mathbf{y}^{i-1}}\}} \tau_i^k(t)\, \eta(s_i^k)} \quad \text{for all } s_i^j \in \mathcal{E}_{\mathbf{y}^{i-1}}. \tag{1}$$

Here $\eta(s_i^j)$ represents the heuristic information, a simple (optional) measure that gives a rough estimate of the *a priori* desirability of adding this component given the current partial solution. $\mathcal{E}_{\mathbf{y}^{i-1}} = \{\alpha \in X_i | (\exists \mathbf{x} \in \mathcal{X}_n^f)(\mathbf{x}_{i]} = \mathbf{y}^{i-1} \oplus \langle \alpha \rangle)\}$ contains the component values that can be used to extend the partial solution $\mathbf{y}^{i-1}$ such that $\mathbf{y}^i$ can still lead to feasible solutions. Consequently, we will denote the construction procedure as *SolutionConstruction*. Using Eq. (1), we have $P(\mathbf{x}|T(t)) = \prod_{i=0}^{n-1} P(\mathbf{x}_{i+1} | \mathbf{x}_{i]}, T(t))$.

Once all ants within one generation have built their solution, each ant will individually update the pheromone trail parameters. In the t-th iteration, K ants construct a set of n-tuples $\mathcal{A}_t$. This set is used in the following update rule:

$$\tau_i^j(t+1) = (1-\rho)\tau_i^j(t) + \rho \frac{\sum_{\{\mathbf{x} \in \mathcal{A}_t | s_i^j \in \mathbf{x}\}} f^*(\mathbf{x})}{\sum_{\{\mathbf{x} \in \mathcal{A}_t\}} f^*(\mathbf{x})}, \tag{2}$$

where $f^*(\mathbf{x})$ is a monotonic transformation of $f(\mathbf{x})$ (often $f^*(\mathbf{x}) = f(\mathbf{x})$). This update is performed for all pheromone trail parameters τ_i^j. As a consequence of this kind of update rule, the pheromones linked with solution components that were part of multiple solutions with high objective function values will receive high updates. Note that Eq. (2) is the update rule used in the hyper-cube framework (HCF) [11]. When the denominator in Eq. (2) is omitted, the standard AS update is obtained. This procedure will be denoted as *PheromoneUpdate*. Combining the principles described above leads to algorithm BasicAS.

```
1:  procedure BasicAS (X_n, f, g)
2:    T(0) ← InitializePheromones
3:    while no convergence do          ▷ terminated if converged
4:      solutionsLastGen ← NULL
5:      for ant = 1, ..., K do         ▷ make paths
6:        x ← SolutionConstruction(T(t))
7:        solutionsLastGen.add(x)
8:      end for
9:      PheromoneUpdate(T(t), solutionsLastGen)
10:   end while
11:   return convergedSolution
12: end procedure
```

3. Bias in metaheuristics

Search bias is a key concept in the field of evolutionary algorithms. Starting from an initial state, evolutionary algorithms are expected to evolve towards the more promising regions in the search space. Typically, these algorithms start searching through the search space in a random, undirected manner. The solutions with the highest objective function values that are found during the first iteration will influence the search direction in the following iteration; they will bias the search towards the point in the search space they represent. This process continues as the solutions that are found during the second iteration will influence the search direction in the third iteration, and so on. As a result, the search might be drawn towards the promising regions in the search space. Stated differently, several positions in the search space will compete for attention from the algorithm. In order to find good solutions, the most attractive points in the search space should have the highest objective function values (and thus be good solutions). Unfortunately, the objective function values are not the only factors that influence the attractiveness of particular regions in the search space. Firstly, as we will see below, the search space can over-represent some solutions, making these solutions more attractive for the search procedure. Secondly, several properties of the evolutionary algorithm that is used can (unwantedly) direct the search towards specific regions in the search space. When such regions represent low-quality solutions, this type of bias can be harmful for an evolutionary algorithm. This phenomenon has been the subject of intensive study in GAs [12–15] and is sometimes referred to as negative search bias. In the field of ACO, negative search bias has drawn some attention as well [2,4,11,16]. In these studies, several (at least three) sources of bias have been identified (we review these sources below). Moreover, [17] define and use a deterministic model to study the dynamics of ACO. More precisely, they use a fixed point analysis of the deterministic model to explain several dynamical properties of ACO.