Search Bias in Ant Colony Optimization: On the Role of Competition-Balanced Systems

Article *in* IEEE Transactions on Evolutionary Computation · May 2005

DOI: 10.1109/TEVC.2004.841688 · Source: IEEE Xplore

2 authors:

Christian Blum
Spanish National Research Council
249 PUBLICATIONS **13,694** CITATIONS
SEE PROFILE

Marco Dorigo
Université Libre de Bruxelles
523 PUBLICATIONS **86,612** CITATIONS
SEE PROFILE

Search Bias in Ant Colony Optimization: On the Role of Competition-Balanced Systems

Christian Blum and Marco Dorigo, *Senior Member, IEEE*

***Abstract*—One of the problems encountered when applying ant colony optimization (ACO) to combinatorial optimization problems is that the search process is sometimes biased by algorithm features such as the pheromone model and the solution construction process. Sometimes this bias is harmful and results in a decrease in algorithm performance over time, which is called *second-order deception*. In this work, we study the reasons for the occurrence of second-order deception. In this context, we introduce the concept of *competition-balanced system* (CBS), which is a property of the combination of an ACO algorithm with a problem instance. We show by means of an example that combinations of ACO algorithms with problem instances that are not CBSs may suffer from a bias that leads to second-order deception. Finally, we show that the choice of an appropriate pheromone model is crucial for the success of the ACO algorithm, and it can help avoid second-order deception.**

***Index Terms*—Algorithm performance, ant colony optimization (ACO), metaheuristics, search bias.**

I. Introduction

METAHEURISTICS, including well-known techniques such as evolutionary algorithms and tabu search, are approximate techniques for the solution of combinatorial optimization problems [5], [21]. In the early 1990s, ant colony optimization (ACO) [13], [16], [17] emerged as a novel nature-inspired metaheuristic. The inspiring source of ACO is the foraging behavior of real ants. When searching for food, ants initially explore the area surrounding their nest in a random manner. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the ant deposits a chemical pheromone trail on the ground. The quantity of pheromone deposited, which may depend on the quantity and quality of the food, will guide other ants to the food source. As it has been shown in [12], indirect communication between the ants via pheromone trails enables them to find shortest paths between their nest and food sources. This characteristic of real ant colonies is exploited in artificial ant colonies in order to solve discrete optimization problems.

While most of the literature on ACO algorithms deals with practical applications [14], [15], [18], recent research efforts have shown the need for a better understanding of the behavior of ACO algorithms. For example, Blum and Sampels [6], [7] studied the application of ACO algorithms to shop scheduling problems. They discovered that the performance of ACO algorithms may decrease over time, depending on the pheromone model and the tackled problem instance. This behavior is clearly undesirable, because in general it worsens the probability of finding better and better solutions over time. Blum *et al.* conducted a related study concerning the k-cardinality tree problem in [8]. In a similar line of work, Merkle and Middendorf [28], [29] studied the behavior of a simple ACO algorithm by analyzing the dynamics of its model when applied to permutation problems. In general, the solution construction process of ACO algorithms for constrained problems consists of a sequence of random decisions, in which later decisions depend on earlier decisions. For ACO algorithms applied to permutation problems the later decisions of the construction process are inherently biased by the earlier ones. In general, the work of Merkle and Middendorf showed that the behavior of ACO algorithms is strongly influenced by the pheromone model (and the way of using it) and by the competition between the ants. As in the work by Blum *et al.* mentioned above, it was also shown that the expected iteration quality of the model of an ACO algorithm may decrease during a run. Recently, Montgomery *et al.* [30] made an attempt to extend the work by Blum and Sampels [6], [7] to assignment problems, and to attribute search bias to different algorithmic components. However, the reasons for the decrease in algorithm performance—which is the subject of this paper—remained largely unknown.

The first attempt of capturing the behavior of ACO algorithms in a formal framework was done in [3] and [4], a work which is closely related to deception in evolutionary computation. The term *deception* was introduced by Goldberg in [22]. Goldberg defined deception in terms of the static average fitness of competing schemas and his aim was to describe problems that are misleading for genetic algorithms (GAs). Since then, much of the work on trying to understand the difficulties that a problem

Manuscript received April 6, 2004; revised October 13, 2004. The work of C. Blum was supported in part by a Fellowship by the Metaheuristics Network under Grant HPRN-CT-1999-00106, a Research Training Network funded by the Improving Human Potential Program of the European Commission, in part by the Spanish CICYT Project under Grant TIC-2002-04498-C05-03 (TRACER), and in part by the Research Training Network "Segravis" under Grant HPRN-CT-2002-00275. The work of M. Dorigo was supported in part by the Belgian FNRS and in part by the ANTS Project, an Action de Recherche Concertée funded by the Scientific Research Directorate of the French Community of Belgium. The work of C. Blum was done while with IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

C. Blum is with the Department of Llenguatges i Sistemes Informátics, Universitat Politècnica de Catalunya, Barcelona E-08034, Spain (e-mail: cblum@lsi.upc.es).

M. Dorigo is with IRIDIA, Université Libre de Bruxelles, Brussels B-01050, Belgium (e-mail: mdorigo@ulb.ac.be).

Digital Object Identifier 10.1109/TEVC.2004.841688

might pose for GAs has focused on deception. Well-known examples of GA-deceptive problems are n-bit trap functions [11]. These functions are characterized by fixpoints with large basins of attraction that correspond to suboptimal solutions, and by fixpoints with relatively small basins of attraction that correspond to optimal solutions. Therefore, for these problems, a GA will—in most cases—not find an optimal solution. In [3] and [4], Blum and Dorigo adopted the term deception for the field of ACO, similarly to what had previously been done in evolutionary computation. It was shown that ACO algorithms, in general, suffer from first-order deception in the same way as GAs do. Blum and Dorigo also introduced the concept of second-order deception, which is caused by a bias that leads to decreasing algorithm performance over time.

Search bias is a subject that is well-studied in the field of evolutionary computation. Research on search bias focused mainly on representational bias, i.e., a bias introduced by the chosen solution representation, and on operator bias, i.e., a bias that is introduced by algorithmic operators such as crossover and mutation. An example for operator bias is the positional bias introduced by one-point crossover when applied to bit-strings [19]. Positional bias refers to the fact that bits that are relatively far apart on an individual will be more likely to be separated by crossover than bits that are close to each other. Conversely, bits that are close to each other are more likely to be treated as units. However, positional bias does not necessarily have a negative impact on the search process. A second example for operator bias is the work by Vekaria and Clack [38], who deal with the bias that is introduced by adaptive recombination operators. They show that this bias is not always beneficial for GA's performance. Operator bias has also been studied in the genetic programming community. For example, the work by Poli *et al.* [34], [35], [26] deals with the biases that are introduced by different crossover operators and mutation sizes. In particular, it was shown that standard subtree crossover induces a very strong bias toward oversampling shorter strings when applied to variable length linear structures.

An example of representational bias arises when the solutions to a problem are the spanning trees of a graph. One possible solution representation in this case is the encoding of the spanning trees by means of Prüfer numbers.[1] Rothlauf and Goldberg [37] dealt with this subject and showed that the Prüfer number encoding introduces a bias toward solutions that have the shape of a star. They were able to show that for problem instances where the optimal solutions are of star shape, their evolutionary algorithm works well and *vice versa*. (Another work on the same subject is the work by Palmer and Kershenbaum [33].)

A second example of negative representational bias can be found in the work by Igel and Stagge [24]. They show that when dealing with redundant encodings, operators that are unbiased in genotype space may exhibit a remarkable bias in phenotype space, which in some cases has a negative impact on the search process.

In general, it was shown that it is important to find solution representations and genetic operators that work well with each other (see, for example, [10] and [39]).

Outline: In this paper, we study the reasons for the occasional decrease of ACO algorithm performance over time, which is labeled second-order deception. Section II introduces the basics of ACO. In Section III, we shortly introduce the framework of first- and second-order deception in ACO. Furthermore, we introduce the concept of *competition-balanced system* (CBS), which is a property of the combination of an ACO algorithm with a problem instance. We show that the property of being a CBS is a positive property for an ACO algorithm/instance combination. As an example, we use the asymmetric traveling salesman problem (ATSP). In Section IV, we study an ACO algorithm for the job shop scheduling (JSS) problem. We show that, in general, combinations of the considered ACO algorithm with JSS instances are not CBSs, and that this may introduce a harmful bias that misleads the search process. Avoiding second-order deception by means of choosing an appropriate pheromone model is studied in Section V. Finally, in Section VI, we offer conclusions and an outlook to future work.

II. Framework of a Basic ACO Algorithm

ACO algorithms are stochastic approximate methods for tackling combinatorial optimization (CO) problems (see [18]). The central component of an ACO algorithm is the pheromone model, which is used to probabilistically sample the search space. As outlined in [3], the pheromone model can be derived from a *model* of the CO problem under consideration. A model of a CO problem can be stated as follows.

Definition 1: A **model** $\mathcal{P} = (\mathcal{S}, \Omega, f)$ of a CO problem consists of the following:

- a **search (or solution) space** $\mathcal{S}$ defined over a finite set of discrete decision variables and a set Ω of **constraints** among the variables;
- an **objective function** $f : \mathcal{S} \rightarrow \mathbb{R}^+$ to be minimized.

The search space $\mathcal{S}$ is defined as follows. Given is a set of n **discrete variables** X_i with values $v_i^j \in D_i = \left\{v_i^1, \ldots, v_i^{|D_i|}\right\}$, $i = 1, \ldots, n$. A variable instantiation, that is, the assignment of a value v_i^j to a variable X_i, is denoted by $X_i = v_i^j$. A feasible solution $s \in \mathcal{S}$ is a complete assignment (i.e., an assignment in which each decision variable has a domain value assigned) that satisfies the constraints. If the set of constraints Ω is empty, then each decision variable can take any value from its domain independently of the values of the other decision variables. In this case, we call $\mathcal{P}$ an **unconstrained** problem model, otherwise, a **constrained** problem model. A feasible solution $s^* \in \mathcal{S}$ is called a **globally optimal solution** (or global optimum), if $f(s^*) \leq f(s) \, \forall \, s \in \mathcal{S}$. The set of globally optimal solutions is denoted by $\mathcal{S}^* \subseteq \mathcal{S}$. To solve a CO problem one has to find a solution $s^* \in \mathcal{S}^*$.

[1] Assuming that the nodes of a completely connected, undirected graph $G = (V, E)$ are labeled from 1 to $|V|$, a Prüfer number is a string of length $|V| - 2$ of node labels that encodes one of the possible spanning trees of G. There is a one-to-one mapping between the possible Prüfer numbers and the spanning trees of a graph.

Algorithm 1: The framework of a basic ACO algorithm
input: An instance P of a CO problem model $\mathcal{P} = (\mathcal{S}, f, \Omega)$.
InitializePheromoneValues($\mathcal{T}$)
$\mathfrak{s}_{bs} \leftarrow$ NULL
while termination conditions not met **do**
  $\mathfrak{S}_{\text{iter}} \leftarrow \emptyset$
  for $j = 1, \ldots, n_a$ **do**
    $\mathfrak{s} \leftarrow$ ConstructSolution($\mathcal{T}$)
    $\mathfrak{s} \leftarrow$ LocalSearch($\mathfrak{s}$) {optional}
    if $(f(\mathfrak{s}) < f(\mathfrak{s}_{bs}))$ or $(\mathfrak{s}_{bs} = \text{NULL})$ **then** $\mathfrak{s}_{bs} \leftarrow \mathfrak{s}$
    $\mathfrak{S}_{\text{iter}} \leftarrow \mathfrak{S}_{\text{iter}} \cup \{\mathfrak{s}\}$
  end for
  ApplyPheromoneUpdate($\mathcal{T}, \mathfrak{S}_{\text{iter}}, \mathfrak{s}_{bs}$)
end while
output: The best-so-far solution $\mathfrak{s}_{bs}$

A model of the CO problem under consideration implies a finite set of solution components and a pheromone model as follows. First, we call the combination of a decision variable X_i and one of its domain values v_i^j a *solution component* denoted by $\mathfrak{c}_i^j$. Then, the pheromone model consists of a *pheromone trail parameter* $\mathcal{T}_i^j$ for each solution component $\mathfrak{c}_i^j$. The set of all solution components is denoted by $\mathfrak{C}$. The value of a pheromone trail parameter $\mathcal{T}_i^j$—called *pheromone value*—is denoted by τ_i^j. The set of all pheromone trail parameters is denoted by $\mathcal{T}$. As a CO problem can be modeled in different ways, different models of a CO problem can be used to define different pheromone models.

Algorithm 1 captures the framework of a basic ACO algorithm. It works as follows. At each iteration, n_a ants probabilistically construct solutions to the combinatorial optimization problem under consideration, exploiting a given pheromone model. Then, optionally, a local search procedure is applied to the constructed solutions. Finally, before the next iteration starts, some of the solutions are used for performing a pheromone update. This framework is explained in more detail in the following.

InitializePheromoneValues($\mathcal{T}$). At the start of the algorithm the pheromone values are all initialized to a constant value $c > 0$.

ConstructSolution($\mathcal{T}$). The basic ingredient of any ACO algorithm is a constructive heuristic for probabilistically constructing solutions. A constructive heuristic assembles solutions as sequences of elements from the finite set of solution components $\mathfrak{C}$. A solution construction starts with an empty partial solution $\mathfrak{s}^p = \langle\rangle$. Then, at each construction step the current partial solution $\mathfrak{s}^p$ is extended by adding a feasible solution component from the set $\mathfrak{N}(\mathfrak{s}^p) \subseteq \mathfrak{C} \setminus \{\mathfrak{s}^p\}$, which is defined by the solution construction mechanism. The process of constructing solutions can be regarded as a walk (or a path) on the so-called *construction graph* $\mathcal{G}_C = (\mathfrak{C}, \mathfrak{L})$, which is a fully connected graph whose vertices are the solution components $\mathfrak{C}$ and the set $\mathfrak{L}$ are the connections. The allowed walks on $\mathcal{G}_C$ are implicitly defined by the solution construction mechanism that defines the set $\mathfrak{N}(\mathfrak{s}^p)$ with respect to a partial solution $\mathfrak{s}^p$. The choice of a solution component $\mathfrak{c}_i^j \in \mathfrak{N}(\mathfrak{s}^p)$ is, at each construction step, done probabilistically with respect to the pheromone model. The probability for the choice of $\mathfrak{c}_i^j$ is proportional to $\left[\tau_i^j\right]^\alpha \cdot \left[\eta\left(\mathfrak{c}_i^j\right)\right]^\beta$, where η is a function that assigns to each valid solution component—possibly depending on the current construction step—a heuristic value which is also called the *heuristic information*. α and β are positive parameters whose values determine the relative importance of pheromone value and heuristic information. The heuristic information is optional, but often needed for achieving a high algorithm performance. In most ACO algorithms, the probabilities for choosing the next solution component—also called the *transition probabilities*—are defined as follows:

$$\mathbf{p}\left(\mathfrak{c}_i^j \mid \mathfrak{s}^p\right) = \frac{\left[\tau_i^j\right]^\alpha \cdot \left[\eta\left(\mathfrak{c}_i^j\right)\right]^\beta}{\sum_{\mathfrak{c}_k^l \in \mathfrak{N}(\mathfrak{s}^p)} \left[\tau_k^l\right]^\alpha \cdot \left[\eta\left(\mathfrak{c}_k^l\right)\right]^\beta} \qquad \forall\, \mathfrak{c}_i^j \in \mathfrak{N}(\mathfrak{s}^p). \tag{1}$$

Note that potentially there are many different ways of choosing the transition probabilities. The above form has mainly historical reasons, because it was used in the first ACO algorithms that were proposed (see, for example, [17]).

LocalSearch($\mathfrak{s}$). Optionally, a local search procedure for improving the solutions constructed by the ants may be applied.

ApplyPheromoneUpdate($\mathcal{T}, \mathfrak{S}_{\text{iter}}, \mathfrak{s}_{bs}$). The aim of the pheromone value update rule is to increase the pheromone values on solution components that have been found in high quality solutions. Most ACO algorithms use a variation of the following update rule:

$$\tau_i^j \leftarrow (1-\rho) \cdot \tau_i^j + \frac{\rho}{n_a} \cdot \sum_{\{\mathfrak{s} \in \mathfrak{S}_{\text{upd}} \mid \mathfrak{c}_i^j \in \mathfrak{s}\}} F(\mathfrak{s}) \tag{2}$$

for $i = 1, \ldots, n$, and $j \in \{1, \ldots, |D_i|\}$. Instantiations of this update rule are obtained by different specifications of $\mathfrak{S}_{\text{upd}}$, which—in all cases that we consider in this paper—is a subset of $\mathfrak{S}_{\text{iter}} \cup \{\mathfrak{s}_{bs}\}$, where $\mathfrak{S}_{\text{iter}}$ is the set of solutions that were constructed in the current iteration, and $\mathfrak{s}_{bs}$ is the best-so-far solution. The parameter $\rho \in (0, 1]$ is called *evaporation rate*. It has the function of uniformly decreasing all the pheromone values. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a suboptimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. $F : \mathfrak{S} \mapsto \mathbb{R}^+$ is a function such that $f(\mathfrak{s}) < f(\mathfrak{s}') \Rightarrow F(\mathfrak{s}) \geq F(\mathfrak{s}'), \forall\, \mathfrak{s} \neq \mathfrak{s}' \in \mathfrak{S}$, where $\mathfrak{S}$ is the set of all the sequences of solution components that may be constructed by the ACO algorithm and that correspond to feasible solutions. $F(\cdot)$ is commonly called the *quality function*. Note that the factor $1/n_a$ is usually not used. We introduce it for the mathematical purpose of studying the expected update of the pheromone values. However, as the factor is constant, it does not change the qualitative behavior of an ACO algorithm.

A well-known example of an instantiation of update rule (2) is the AS-update rule, that is, the update rule of ant system (AS) [17]. The AS-update rule is obtained from update rule (2) by setting

$$\mathfrak{S}_{\text{upd}} \leftarrow \mathfrak{S}_{\text{iter}}. \tag{3}$$

This update rule is well-known due to the fact that AS was the first ACO algorithm to be proposed in the literature. An example of a pheromone update rule that is more used in practice is the IB-update rule (where IB stands for Iteration-Best). The IB-update rule is given by

$$\mathfrak{S}_{\text{upd}} \leftarrow \text{argmax}\{F(\mathfrak{S}) \mid \mathfrak{s} \in \mathfrak{S}_{\text{iter}}\}. \tag{4}$$

The IB-update rule introduces a much stronger bias toward the good solutions found than the AS-update rule. However, this increases the danger of premature convergence.

III. Deception in Ant Colony Optimization

The desired behavior of an ACO algorithm can be stated as follows: The average quality of the generated solutions should improve over time. This is desirable, as it usually increases the probability of improving the best solution found over time. First and second-order deception in ACO were defined in [4] as properties of *models* of ACO algorithms, which are deterministic dynamical systems obtained by applying the expected pheromone update instead of the real pheromone update (see, for example, [29]). The advantage of defining the behavior of ACO algorithms in terms of properties of their models is that a model—as it is deterministic—always behaves in the same, in contrast to the behavior of the ACO algorithm itself which in each run slightly differs due to the stochasticity. An ACO algorithm model can be characterized by the evolution of the expected quality of the solutions that are generated per iteration. This expected iteration quality is henceforth denoted by $W_F(\mathcal{T})$, or by $W_F(\mathcal{T} \mid t)$, where $t > 0$ is the iteration counter.

As an example of an ACO algorithm model, let us consider the AS algorithm, which is characterized by the AS-update rule as outlined in the previous section. Assuming n_a solution constructions per iteration, the expected iteration quality is given by

$$W_F(\mathcal{T}) = \sum_{S^{n_a} \in \mathfrak{S}^{n_a}} \left(\mathbf{p}(S^{n_a} \mid \mathcal{T}) \cdot \frac{1}{n_a} \cdot \sum_{\mathfrak{s} \in S^{n_a}} F(\mathfrak{s}) \right) \tag{5}$$

where $\mathfrak{S}^{n_a}$ is the set of all multisets of cardinality n_a consisting of elements from $\mathfrak{S}$, and $\mathbf{p}(S^{n_a} \mid \mathcal{T})$ is the probability that the n_a ants produce the multiset $S^{n_a} \in \mathfrak{S}^{n_a}$ of solutions. The expected pheromone update of the AS algorithm is then obtained by setting $\tau_i^j(t+1)$ to

$$(1-\rho) \cdot \tau_i^j(t) + \frac{\rho}{n_a} \cdot \sum_{S^{n_a} \in \mathfrak{S}^{n_a}} \left(\mathbf{p}(S^{n_a} \mid \mathcal{T}) \cdot \sum_{\{\mathfrak{s} \in S^{n_a} \mid \mathfrak{c}_i^j \in \mathfrak{s}\}} F(\mathfrak{s}) \right) \tag{6}$$

for $i = 1, \ldots, n$, $j = 1, \ldots, |D_i|$ (note that j depends on i), and where t is the iteration counter.

In order to reduce the computational complexity of this model, we simplify it by assuming an infinite number of solution constructions (i.e., ants) per iteration (henceforth, we will refer to this as to the *simplified model*). In this case, the expected iteration quality is given by

$$W_F(\mathcal{T}) = \sum_{\mathfrak{s} \in \mathfrak{S}} F(\mathfrak{s}) \cdot \mathbf{p}(\mathfrak{s} \mid \mathcal{T}) \tag{7}$$

where $\mathbf{p}(\mathfrak{s} \mid \mathcal{T})$ is the probability to produce solution $\mathfrak{s}$ given the current pheromone values. The expected AS pheromone update is given by

$$\tau_i^j(t+1) \leftarrow (1-\rho) \cdot \tau_i^j(t) + \rho \cdot \sum_{\{\mathfrak{s} \in \mathfrak{S} \mid \mathfrak{c}_i^j \in \mathfrak{s}\}} F(\mathfrak{s}) \cdot \mathbf{p}(\mathfrak{s} \mid \mathcal{T}). \tag{8}$$

Definition 2: Given a model $\mathcal{P}$ of a CO problem, we call a model of an ACO algorithm applied to any instance P of $\mathcal{P}$ a **local optimizer** if for any initial setting of the pheromone values the expected update of the pheromone values is such that $W_F(\mathcal{T} \mid t+1) \geq W_F(\mathcal{T} \mid t), \forall\, t \geq 0$. In other words, the expected quality of the generated solutions per iteration must be monotonically nondecreasing.

The definition of a first-order deceptive system (FODS) is based upon the definition of a local optimizer.

Definition 3: Given a model $\mathcal{P}$ of a CO problem, we call a local optimizer **applied to** instance P of $\mathcal{P}$ a **FODS** if an initial setting of the pheromone values exists such that the local optimizer does not converge to a globally optimal solution.

This means that even if a model of an ACO algorithm is a local optimizer, it is a FODS when, for example, it is applied to problem instances that are characterized by the fact that they induce more than one stable fixpoint.[2] An example is the application of the simplified model of AS to n-bit trap functions (see[3]).

However, first-order deception does not constitute a major problem in ACO. In fact, it is a desirable property of any optimization algorithm that the average quality of the generated solutions increases over time. Therefore, we are generally satisfied if a model of an algorithm is a local optimizer. As a matter of fact, metaheuristics are in general not global optimizers. In contrast, if it cannot be shown that the considered model of an algorithm is a local optimizer, the empirical behavior of the algorithm might be very difficult to predict. Therefore, a second-order deceptive system (SODS) is defined as a system that does not have the property of being a local optimizer.

Definition 4: Given a model $\mathcal{P}$ of a CO problem, we call a model of an ACO algorithm **applied to** instance P of $\mathcal{P}$ a **SODS**, if the evolution of the expected iteration quality contains time windows $[i, i+l]$ (where $i > 0$, $l > 0$) with $W_F(\mathcal{T} \mid t+1) < W_F(\mathcal{T} \mid t), \forall\, t \in \{i, \ldots, i+l-1\}$. This means that the combination of a model of an ACO algorithm and an instance P of $\mathcal{P}$ may be a SODS, if the ACO algorithm model is not a local optimizer. We henceforth refer to the above mentioned time windows as **second-order deception effects**. Also, with respect to the empirical behavior of an ACO algorithm, we will use the notion of second-order deception effects.

In the remainder of this paper, we will examine the question of when second-order deception effects can be expected to

[2]The term of a stable fixpoint stems from the field of discrete dynamical systems. It denotes the state of a system that is characterized by the fact that when the system has entered this state it will never leave it (see, for example, [23]).

arise. For this purpose, we examine more closely the relation of the solution components with the solution construction process. Each step of constructing a solution is a probabilistic decision on how to extend the current partial solution. At each of these construction steps, solution components compete with each other. From this point of view, the dynamics of an ACO algorithm can be described as the competition between the different solution components. This competition should of course, be as fair as possible. The following definition formally characterizes a fair competition between the solution components.

Definition 5: Given a model $\mathcal{P}$ of a CO problem, we call the combination of an ACO algorithm and a problem instance P of $\mathcal{P}$ a **CBS**, if the following holds: Given a feasible partial solution $\mathfrak{s}^p$ and the set of solution components $\mathfrak{N}(\mathfrak{s}^p)$ that can be added to extend $\mathfrak{s}^p$, each solution component $\mathfrak{c} \in \mathfrak{N}(\mathfrak{s}^p)$ is a component of the same number of feasible solutions (in terms of sequences built by the algorithm) as any other solution component $\mathfrak{c}' \in \mathfrak{N}(\mathfrak{s}^p)$, $\mathfrak{c} \neq \mathfrak{c}'$.[3] In this context, we call the competition between the solution components a **fair competition** if the combination of an ACO algorithm and a problem instance is a CBS.

In order to show that a fair competition between the solution components most likely results in a search process that does not suffer from a harmful bias, we applied the simplified model of the AS algorithm as outlined above to the asymmetric traveling salesman problem (ATSP) [25]. The pheromone model and the solution construction process that we used are the standard ones for applications of ACO to traveling salesman problems (TSPs) (see [17]). The solution components correspond to links in the ATSP graph, and each link is associated with a pheromone value. The AS algorithm applied to any ATSP problem instance is a CBS. This is because each link (i.e., each solution components) is in the same number of solutions (i.e., Hamiltonian cycles in the ATSP graph), namely, $(n-1)!$ solutions. Therefore, at each construction step the competing solution components are in the same number of feasible solutions.

We applied the simplified model of AS to 10^3 randomly generated ATSP instances of size $n \in \{3, \ldots, 10\}$. Note that for applying this model, the search space must be enumerated. Therefore, we had to restrict ourselves to problem instances of small size. The initial value of each of the pheromone trail parameters was randomly chosen from $\{1, \ldots, 100\}$. We stopped the algorithm for each of the problem instances once the change in expected iteration quality from one iteration to the other was smaller than 10^{-6}. In all 10^3 experiments, we did not find a single case of decreasing expected iteration quality from one iteration to the other. Fig. 1 shows the evolution of the expected iteration quality of the simplified model of AS when applied to a randomly generated ATSP instance of size $n = 7$ for different settings of ρ. Our results suggest that the simplified model of AS for the ATSP is a local optimizer, i.e., it is not a SODS. We also conjecture that this is largely determined by the property of being a CBS. In Section IV, we show on the example of the JSS problem how the lack of this property can lead to algorithms that suffer from second-order deception effects.

[3]Note that there exist ACO algorithms in which partial solutions are extended by "groups" of solution components. In these cases, the definition of a CBS has to be adapted accordingly.

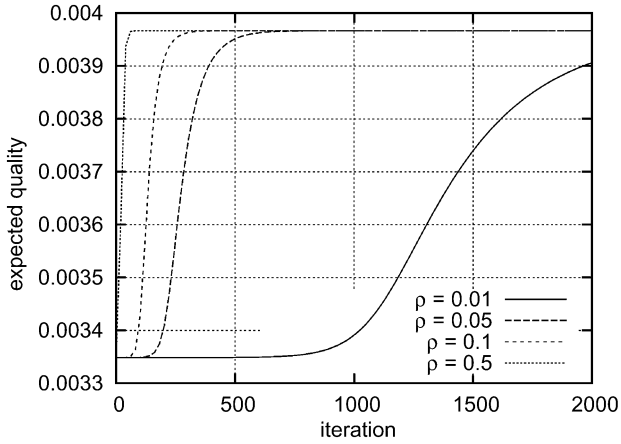


Fig. 1. Evolution of the expected iteration quality W_F of the simplified model of AS applied to a randomly generated ATSP instance on seven nodes (cities). The four plots show the evolution of the expected iteration quality over time for different settings of parameter ρ. Each initial pheromone value was randomly chosen from $\{1, \ldots, 100\}$. The plots show that the expected iteration quality continuously increases. Moreover, for increasing ρ the impact of the pheromone value update increases, because the algorithm is less conservative.

IV. Example of Second-Order Deception

Three examples of second-order deception can be found in the literature [7], [8], [29]. In [29], Merkle and Middendorf discovered a case of second-order deception when studying a model of a simple ACO algorithm for permutation problems. In [8], Blum *et al.* showed second-order deception in the context of an ACO algorithm applied to the node-weighted k-cardinality tree (KCT) problem. However, in this case second-order deception can only be detected when the AS-update rule is applied without the use of local search for improving the solutions constructed by the ants. A more serious case of second-order deception is reported by Blum and Sampels in [6] and [7] for the $\mathcal{NP}$-hard JSS [2] problem.

In the JSS problem, we are given a set of operations $\mathcal{O} = \{o_1, \ldots, o_n\}$, which is partitioned into a set of subsets $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_{|\mathcal{M}|}\}$. The operations in $\mathcal{M}_i \in \mathcal{M}$ have to be processed on the same machine. For the sake of simplicity, we identify each set $\mathcal{M}_i \in \mathcal{M}$ of operations with the machine they have to be processed on, and call $\mathcal{M}_i$ a machine. $\mathcal{O}$ is also partitioned into a set of subsets $\mathcal{J} = \{\mathcal{J}_1, \ldots, \mathcal{J}_{|\mathcal{J}|}\}$, where the set of operations $\mathcal{J}_j \in \mathcal{J}$ is called a job. Furthermore, a processing time $p(o) \in \mathbb{N}$ is given for each operation $o \in \mathcal{O}$. Also given are permutations $\pi^{\mathcal{J}_j}$ of the operations of each job $\mathcal{J}_j \in \mathcal{J}$. Such a job-permutation defines the order in which the operations of the corresponding job have to be processed.[4] In the following, when given a permutation π, $\pi(l)$ denotes the operation on the lth position of permutation π. A permutation of the operations of a job $\mathcal{J}_j \in \mathcal{J}$ imposes a total order $\preceq$ on the operations of $\mathcal{J}_j$ and a partial order on $\mathcal{O}$. For $l < k$ and the two operations $o_i = \pi(l)$ and $o_j = \pi(k)$, we write $o_i \preceq o_j$. The set of *predecessors* of an operation $o_i \in \mathcal{O}$ is given by

$$\mathrm{pred}(o_i) \leftarrow \{o_j \in \mathcal{O} \mid o_j \preceq o_i\}. \tag{9}$$

[4]The job-permutations are often called technological sequences.

We consider the case in which each machine can process at most one operation at a time. Operations must be processed without preemption (that is, once the processing of an operation has started it must be completed without interruption). Operations belonging to the same job must be processed sequentially.

A solution $s = (\pi^{\mathcal{M}_1}, \ldots, \pi^{\mathcal{M}_{|\mathcal{M}|}})$ is given by permutations $\pi^{\mathcal{M}_i}$ of the operations in $\mathcal{M}_i, \forall\, i \in \{1, \ldots, |\mathcal{M}|\}$. These permutations define processing orders on all the machines $\mathcal{M}_i$. Note that not all combinations of permutations are feasible, because some combinations of permutations might define cycles in the processing orders.

There are several possibilities to measure the cost of a solution. Here, we deal with makespan minimization for which the objective function is defined as follows. Every operation $o \in \mathcal{O}$ has a well-defined *earliest starting time* $t_{es}(o, s)$ with respect to a solution s. We assume that the operations being the first ones to be processed on their machine, as well as in their job, have an earliest starting time of 0. The *earliest completion time* of an operation $o \in \mathcal{O}$ with respect to a solution s is denoted by $t_{ec}(o, s)$ and defined as $t_{es}(o, s) + p(o)$. Then, given a solution s, the objective function value (also called the makespan) is defined as

$$f(s) \leftarrow \max\{t_{ec}(o, s) \mid o \in \mathcal{O}\}. \tag{10}$$

The objective is to find a solution with minimum makespan.

Given a JSS problem instance, let us consider the set of all permutations of all operations. Some of these permutations represent solutions to the given JSS instance. This is due to the fact that a permutation of all the operations contains the machine-permutations.[5] Based on this observation, Colorni *et al.* [9] proposed the following model of the JSS problem: First, the set of operations $\mathcal{O}$ is augmented by two dummy operations o_0 and o_{n+1} with processing time zero. Operation o_0 serves as source operation and o_{n+1} as destination operation. The augmented set of operations is $\mathcal{O} = \{o_0, o_1, \ldots, o_n, o_{n+1}\}$. Then, for each operation o_i, where $i \in \{0, \ldots, n\}$, a decision variable X_i is introduced. The domain for decision variable X_0 is $D_0 = \{1, \ldots, n\}$, and for every other decision variable X_i the domain is $D_i = \{1, \ldots, n+1\} \setminus \{i\}$. The meaning of a domain value $j \in D_i$ for a decision variable X_i is that operation o_j is placed immediately after operation o_i in the permutation of all the operations under construction. This problem model is, henceforth, denoted by $\mathcal{P}^{\text{suc}}_{\text{JSS}}$. In order to derive the pheromone model, we again introduce for each combination of a decision variable X_i and a domain value $j \in D_i$ a solution component $\mathfrak{c}_i^j$. The pheromone model $\mathcal{T}$ then consists of a pheromone trail parameter $\mathcal{T}_i^j$ for each solution component $\mathfrak{c}_i^j$.

Many constructive mechanisms build solutions in terms of feasible permutations to JSS instances by constructing a permutation from left to right. In the following, we refer to this type of algorithm as a *list scheduler algorithm*.[6] A list scheduler algorithm builds a sequence of all operations from left to right. This is done by appending at each of n construction steps to the respective partial solution an operation from set $\mathcal{O}_t$, which is the set of allowed operations. Set $\mathcal{O}_t$ is defined as follows. At each step $t \in \{1, \ldots, n\}$ the set $\mathcal{O}$ of operations is partitioned into set $\mathcal{O}^-$, the set of operations that are already in the partial sequence, and set $\mathcal{O}^+$, the set of operations that still have to be dealt with. However, in order to exclusively generate feasible solutions, $\mathcal{O}_t$ is defined as a subset of $\mathcal{O}^+$ in the following way[7]:

$$\mathcal{O}_t \leftarrow \{o \in \mathcal{O}^+ \mid \text{pred}(o) \cap \mathcal{O}^+ = \emptyset\}. \tag{11}$$

The solution construction mechanism of the ACO algorithm by Colorni *et al.* [9] uses the mechanism of list scheduler algorithms and works as follows. Let i_c denote the index of the decision variable that receives a value in the current construction step. Further, at each constructions step index set $\mathcal{I}$ contains i_c as well as the indexes of the decision variables that have already assigned a value. The solution construction starts with an empty partial solution $\mathfrak{s}^p = \langle\rangle$, with $i_c = 0$, and with $\mathcal{I} = \{0\}$. Then, at each of n construction steps $t = 1, \ldots, n$, a solution component $\mathfrak{c}_{i_c}^j \in \mathfrak{N}(\mathfrak{s}^p)$ is added to the current partial solution, where

$$\mathfrak{N}(\mathfrak{s}^p) = \left\{\mathfrak{c}_{i_c}^k \mid o_k \in \mathcal{O}_t\right\}. \tag{12}$$

This means that at each construction step we choose a domain value for the decision variable with index i_c. When adding the solution component $\mathfrak{c}_{i_c}^j$ to $\mathfrak{s}^p$, we also set i_c to j and add j to $\mathcal{I}$. In the $(n+1)$th construction step, the value of the last unassigned decision variable X_{i_c} is set to $n+1$.

Each construction step is done according to the following probability distribution, which is the same as (1), except that we do not consider heuristic information:

$$\mathbf{p}(\mathfrak{c}_{i_c}^j \mid \mathcal{T}) = \frac{\tau_{i_c}^j}{\sum_{\mathfrak{c}_{i_c}^k \in \mathfrak{N}(\mathfrak{s}^p)} \tau_{i_c}^k} \qquad \forall\, \mathfrak{c}_{i_c}^j \in \mathfrak{N}(\mathfrak{s}^p). \tag{13}$$

The pheromone value update rule to be examined is again the AS-update rule given in (2) and (3). The instantiation of Algorithm 1 as defined above by the definition of the pheromone model, the solution construction mechanism and the pheromone update rule is, henceforth, denoted by AS_JSS_suc.

A. Study of a Small Problem Instance

The small problem instance that we consider in the following is specified as follows:

$$\begin{aligned} \mathcal{O} &= \{o_1, o_2, o_3, o_4\} \\ \mathcal{J} &= \{\mathcal{J}_1 = \{o_1, o_2\}, \mathcal{J}_2 = \{o_3, o_4\}\} \\ \mathcal{M} &= \{\mathcal{M}_1 = \{o_1, o_4\}, \mathcal{M}_2 = \{o_2, o_3\}\} \\ p(o_1) &= p(o_4) = 10 \\ p(o_2) &= p(o_3) = 20. \end{aligned} \tag{14}$$

We, henceforth, denote this problem instance by jss_simple_inst. An example of the construction of a solution to this problem instance is shown in Fig. 2(a). In Fig. 2(b), which shows all possible solution constructions in

[5]However, note that there is generally a many-to-one mapping from the set of feasible permutations of n operations to the set of different solutions to a JSS problem instance.

[6]One of the most prominent list scheduler algorithms is the GT algorithm [20]. Note that in the literature there is no well-established term for this type of algorithm. They are also often referred to as list scheduling algorithms, priority rule heuristics, or priority rule systems.

[7]Remember that $\text{pred}(o)$ is the set of predecessors of operation o as defined in (9).

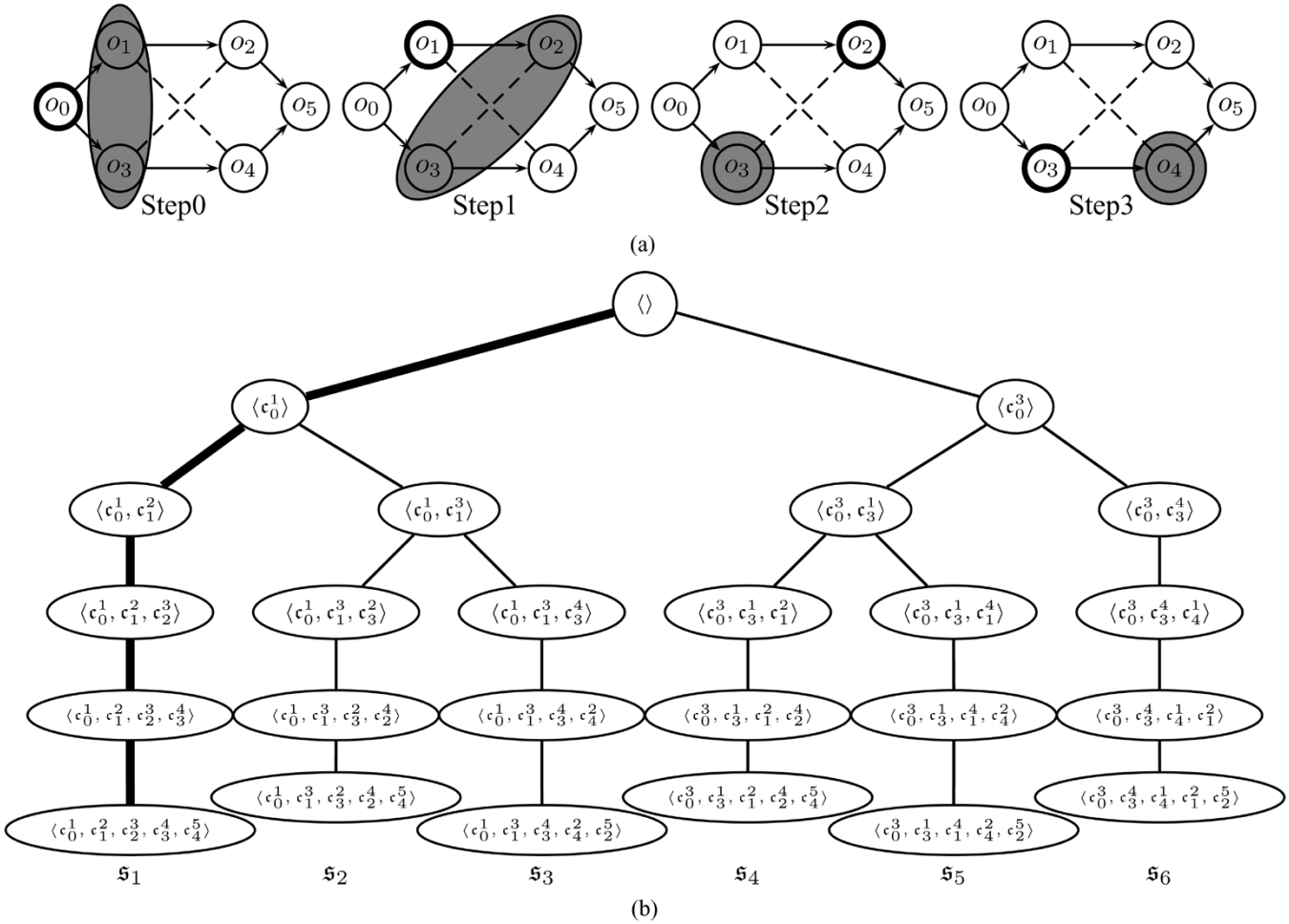


Fig. 2. (a) Construction of solution $\mathfrak{s}_1 = \langle \mathfrak{c}_0^1, \mathfrak{c}_1^2, \mathfrak{c}_2^3, \mathfrak{c}_3^4, \mathfrak{c}_4^5 \rangle$ to problem instance jss_simple_inst as specified in (14). Solution $\mathfrak{s}_1$ corresponds to solution $s_1 = \{X_0 = 1,\ X_1 = 2,\ X_2 = 3,\ X_3 = 4,\ X_4 = 5\}$ to the CO problem model $\mathcal{P}_{\text{JSS}}^{\text{suc}}$. At each step, the operation that is shown in bold face is the current operation. The operations covered by the gray shaded areas are the allowed values for the decision variable that corresponds to the current operation. (b) Complete search tree as given by the solution construction mechanism of AS_JSS_suc applied to problem instance jss_simple_inst. The path in the search tree that corresponds to the solution construction that is shown in (a) is indicated in bold.

form of a search tree, it can be seen that AS_JSS_suc may produce six different sequences

$$\begin{aligned}
\mathfrak{s}_1 &= \langle \mathfrak{c}_0^1, \mathfrak{c}_1^2, \mathfrak{c}_2^3, \mathfrak{c}_3^4, \mathfrak{c}_4^5 \rangle \\
\mathfrak{s}_2 &= \langle \mathfrak{c}_0^1, \mathfrak{c}_1^3, \mathfrak{c}_3^2, \mathfrak{c}_2^4, \mathfrak{c}_4^5 \rangle \\
\mathfrak{s}_3 &= \langle \mathfrak{c}_0^1, \mathfrak{c}_1^3, \mathfrak{c}_3^4, \mathfrak{c}_4^2, \mathfrak{c}_2^5 \rangle \\
\mathfrak{s}_4 &= \langle \mathfrak{c}_0^3, \mathfrak{c}_3^1, \mathfrak{c}_1^2, \mathfrak{c}_2^4, \mathfrak{c}_4^5 \rangle \\
\mathfrak{s}_5 &= \langle \mathfrak{c}_0^3, \mathfrak{c}_3^1, \mathfrak{c}_1^4, \mathfrak{c}_4^2, \mathfrak{c}_2^5 \rangle \\
\mathfrak{s}_6 &= \langle \mathfrak{c}_0^3, \mathfrak{c}_3^4, \mathfrak{c}_4^1, \mathfrak{c}_1^2, \mathfrak{c}_2^5 \rangle
\end{aligned} \tag{15}$$

that map one-to-one to solutions to the CO problem model $\mathcal{P}_{\text{JSS}}^{\text{suc}}$

$$\begin{aligned}
\mathfrak{s}_1 &\mapsto s_1 = \{X_0 = 1, X_1 = 2, X_2 = 3, X_3 = 4, X_4 = 5\} \\
\mathfrak{s}_2 &\mapsto s_2 = \{X_0 = 1, X_1 = 3, X_2 = 4, X_3 = 2, X_4 = 5\} \\
\mathfrak{s}_3 &\mapsto s_3 = \{X_0 = 1, X_1 = 3, X_2 = 5, X_3 = 4, X_4 = 2\} \\
\mathfrak{s}_4 &\mapsto s_4 = \{X_0 = 3, X_1 = 2, X_2 = 4, X_3 = 1, X_4 = 5\} \\
\mathfrak{s}_5 &\mapsto s_5 = \{X_0 = 3, X_1 = 4, X_2 = 5, X_3 = 1, X_4 = 2\} \\
\mathfrak{s}_6 &\mapsto s_6 = \{X_0 = 3, X_1 = 2, X_2 = 5, X_3 = 4, X_4 = 1\}.
\end{aligned} \tag{16}$$

Furthermore, the six solutions to the CO model correspond to three different solutions in terms of sets of machine-permutations

$$\begin{aligned}
s_1 \simeq (1,2,3,4) &\mapsto \left(\pi^{\mathcal{M}_1} = (1,4), \pi^{\mathcal{M}_2} = (2,3)\right) \\
\left.\begin{array}{l} s_2 \simeq (1,3,2,4) \\ s_3 \simeq (1,3,4,2) \\ s_4 \simeq (3,1,2,4) \\ s_5 \simeq (3,1,4,2) \end{array}\right\} &\mapsto \left(\pi^{\mathcal{M}_1} = (1,4), \pi^{\mathcal{M}_2} = (3,2)\right) \\
s_6 \simeq (3,4,1,2) &\mapsto \left(\pi^{\mathcal{M}_1} = (4,1), \pi^{\mathcal{M}_2} = (3,2)\right)
\end{aligned} \tag{17}$$

where the sign $\simeq$ is to be read as "corresponds to."

The objective function values are as follows: $f(\mathfrak{s}_i) = 60$, for $i = 1, 6$, and $f(\mathfrak{s}_i) = 40$, for $i = 2, 3, 4, 5$. Therefore, $\mathfrak{s}_1$ and $\mathfrak{s}_6$ are suboptimal solutions, whereas the other four solutions [which map in fact to the same set of machine-permutations, as shown in (17)] are optimal. Examining the search tree that is shown in Fig. 2(b) reveals that AS_JSS_suc applied to problem instance jss_simple_inst is not a CBS (see Definition 5). An example is the construction step with $\mathfrak{s}^p = \langle \mathfrak{c}_0^1 \rangle$ as the current partial solution. The two solution components that can be added to this current partial solution are $\mathfrak{c}_1^2$ and $\mathfrak{c}_1^3$.

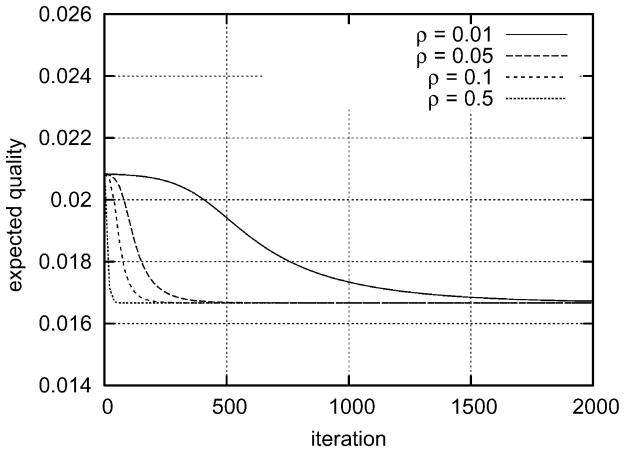


Fig. 3. Evolution of the expected iteration quality W_F of the simplified model of AS_JSS_suc applied to problem instance jss_simple_inst as specified in (14) for different settings of parameter ρ. All the pheromone values were initially set to 0.5. The plots show that the expected iteration quality continuously decreases. Moreover, for increasing ρ the impact of the pheromone value update increases and the average iteration quality decreases faster.

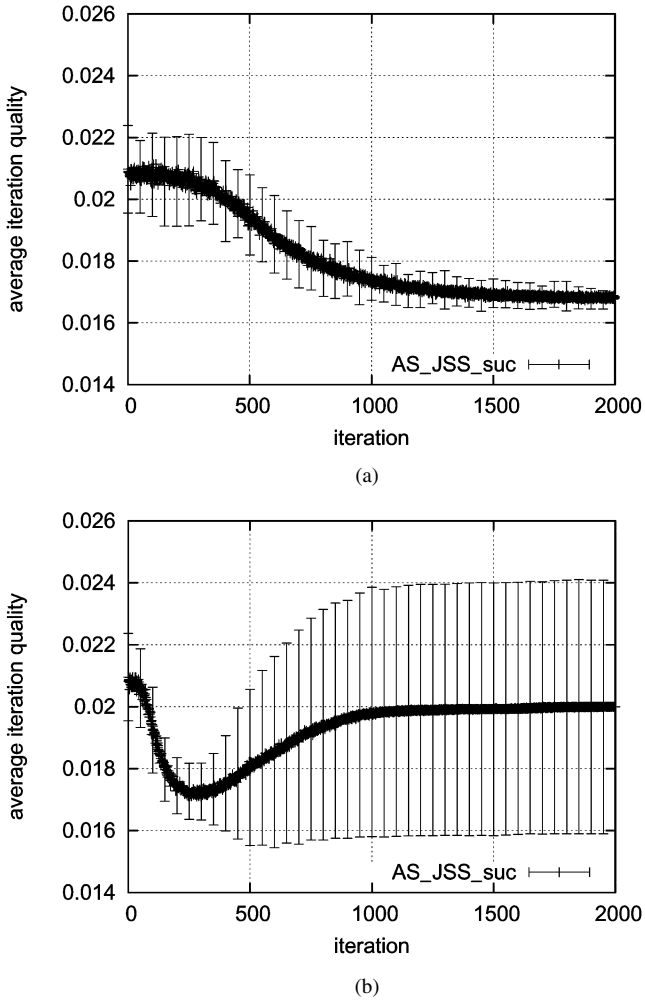


Fig. 4. Plots show the evolution of the average iteration quality obtained by AS_JSS_suc applied to problem instance jss_simple_inst as specified in (14) for $n_a = 10$ and two different settings of parameter ρ (i.e., (a) with $\rho = 0.01$, and (b) with $\rho = 0.05$). All the pheromone values were initially set to 0.5. The results are averaged over 100 runs (the error bars that show the standard deviation are shown every 50th iteration). Note that in (b) the standard deviation of the algorithm behavior increases over time, because the algorithm—due to a higher learning rate—converges sometimes to a good solution and sometimes to a bad solution.

However, solution component $\mathfrak{c}_1^2$ is part of the three solutions $\mathfrak{s}_1$, $\mathfrak{s}_4$ and $\mathfrak{s}_6$, whereas solution component $\mathfrak{c}_1^3$ is only part of the two solutions $\mathfrak{s}_2$ and $\mathfrak{s}_3$. This means that the value of the pheromone trail parameter $\mathcal{T}_1^2$ on average receives updates from one optimal solution (with quality $1/40$) and two suboptimal solutions (with quality $1/60$ each), whereas the value of $\mathcal{T}_1^3$ on average receives updates from two optimal solutions (with quality $1/40$ each). This means that for many initial settings of the pheromone values (e.g., pheromone values all initialized to the same value $c > 0$) $\mathcal{T}_1^2$ on average receives a higher amount of pheromone because the number of solutions that contribute to the updates is higher than that for $\mathcal{T}_1^3$. For symmetry reasons, the same holds for pheromone trail parameters $\mathcal{T}_3^4$ (corresponding to $\mathcal{T}_1^2$) and $\mathcal{T}_3^1$ (corresponding to $\mathcal{T}_1^3$). Therefore, over time, the probability of constructing the suboptimal solutions $\mathfrak{s}_1$ and $\mathfrak{s}_6$ increases, whereas the probability of constructing the optimal solutions $\mathfrak{s}_i$, for $i = 2, \ldots, 5$, decreases. This means that the expected iteration quality decreases over time.

We implemented AS_JSS_suc with the aim of confirming our theoretical considerations and in order to compare the evolution of the empirically obtained average iteration quality of AS_JSS_suc with the evolution of the expected iteration quality W_F of the simplified model of AS_JSS_suc over time. Fig. 3 shows that—for any of the chosen evaporation rates—W_F continuously decreases. The empirical behavior—as shown in Fig. 4—approximates the expected behavior for small ρ and, therefore, shows the second-order deception effects that are caused (as explained above) by the fact that the system is not a competition-balanced one.

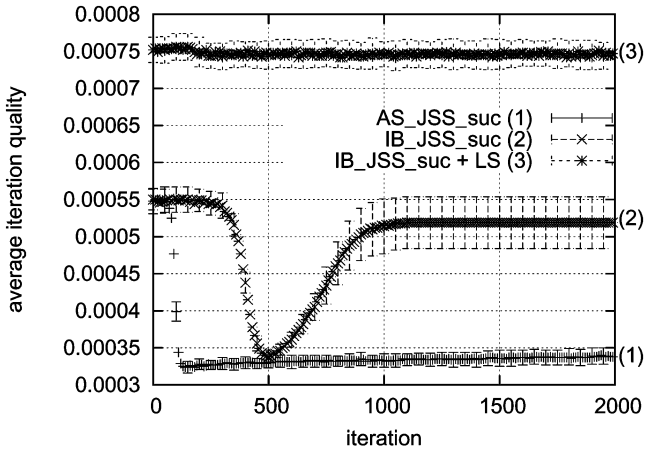
B. Study of Benchmark Instances

Our aim in this section is twofold. First, we show that the results that we obtained in the previous section also hold when 1) large benchmark instances are considered and 2) when practically relevant ACO algorithms (i.e., using the IB-update rule and local search[8] for improving the ant solutions) are used. Second, we show that the harmful bias is due to the fact that, in general, the combination of the above explained ACO algorithm with JSS instances is not a CBS. In the following, we refer to the ACO algorithm that uses the IB-update rule as IB_JSS_suc. We chose the following two problem instances as test cases.

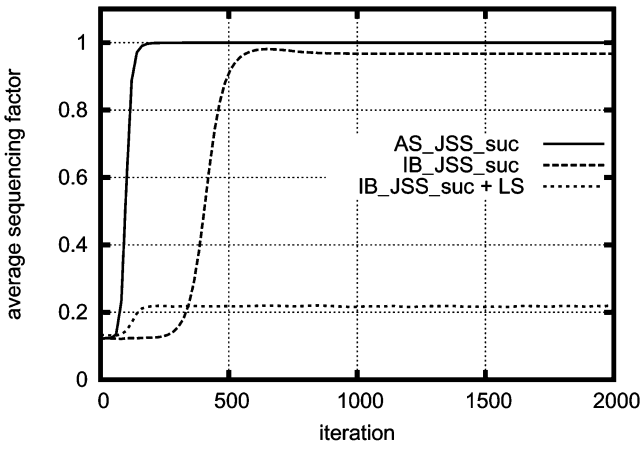
1) $ft10$: This problem instance consists of 100 operations (i.e., ten jobs and ten machines). It was introduced in [31] and is one of the most famous JSS benchmark instances as it remained unsolved for more than 25 years.
2) $orb08$: This second benchmark instance consists likewise of 100 operations (i.e., ten jobs and ten machines). It was introduced by Applegate and Cook in [1].

The results of three versions of our ACO algorithm—namely, AS_JSS_suc, IB_JSS_suc, and IB_JSS_suc + local

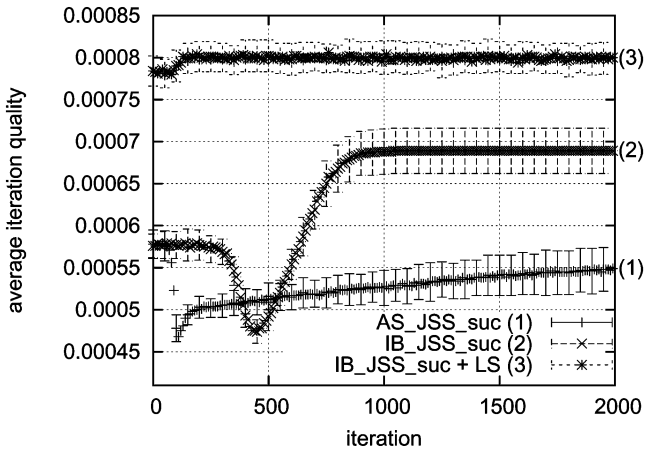
[8]We used the steepest descent local search based on the neighborhood structure introduced by Nowicki and Smutnicki in [32].



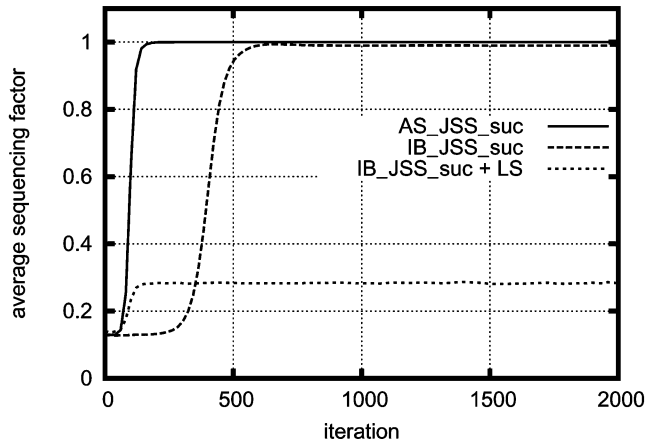
(a) Instance $ft10$, performance.



(b) Instance $ft10$, sequencing factors.



(c) Instance $orb08$, performance.

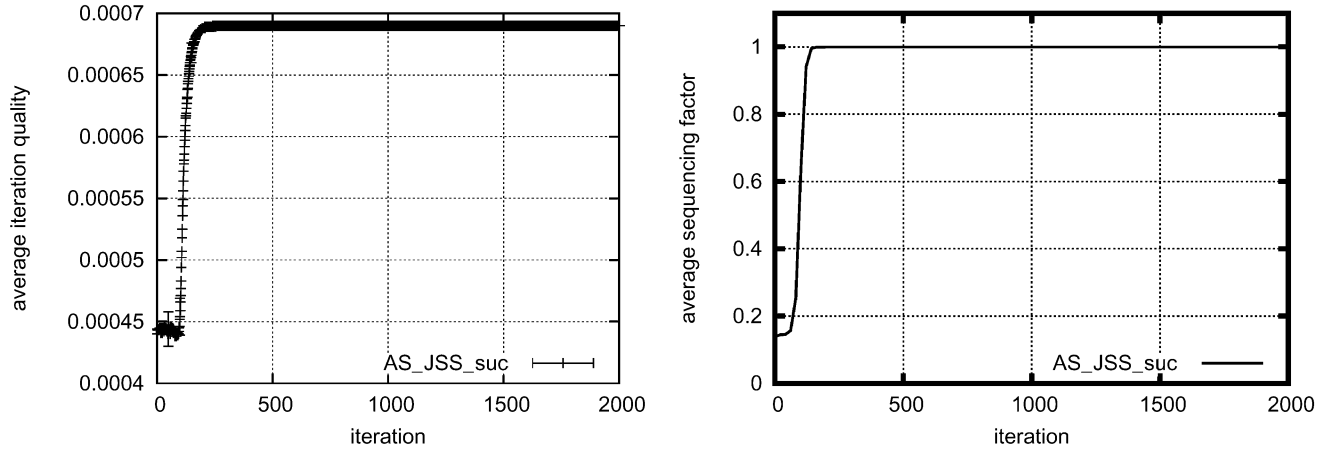


(d) Instance $orb08$, sequencing factors.

Fig. 5. (a) and (c) Shows the evolution of the average iteration quality obtained by three algorithm versions—namely, AS_JSS_suc, IB_JSS_suc, and IB_JSS_suc using local search—when applied to problem instances $ft10$ and $orb08$ for 2000 iterations with $n_a = 10$ and with the evaporation rate ρ optimally chosen (in an experimental way). All the pheromone values were initially set to 0.5. The results are averaged over 100 runs (the error bars that show the standard deviation are shown every 50th iteration). (b) and (d) Shows the evolution of the average sequencing factors of the iteration-best solutions corresponding to the experiments in (a) and (c).

search (LS)—are shown in Fig. 5(a) and (c). Concerning AS_JSS_suc, we can observe for both instances a strong decrease in average iteration quality for the first approximately 150 iterations. Then, the curve of the average iteration quality does a sharp upturn. The average iteration quality of AS_JSS_suc starts to increase slightly in the case of $ft10$ and more strongly in the case of $orb08$. However, even though the average iteration quality increases, the system does not even reach the average quality of the random solutions that were produced in the first iterations, for either problem instance. The results also show that the use of the IB-update reduces the effect of the harmful bias. The second-order deception effects are delayed, and the increase in average iteration quality, after the bottom point in terms of average iteration quality is reached, is stronger. In the case of instance $ft10$, the average iteration quality that is reached at the end of the algorithm run is still lower than the average iteration quality obtained in the first iteration. Finally, when IB_JSS_suc is used in conjunction with local search for improving the solutions constructed by the ants, the algorithm does not seem to be able to learn much, i.e., the average iteration quality does not change notably during the whole algorithm run. In this case, the two forces, i.e., the harmful bias on one side and the forces of the algorithm such as the pheromone update and the local search algorithm on the other side, seem to neutralize each other. Summarizing, our results show that second-order deception is not only an issue for small and opportunely defined JSS problem instances.

Our second goal is to explore the nature of the harmful bias that we identified. An examination of the small example instance jss_simple_inst introduced in (14) reveals the following. Solution $\mathfrak{s}_1$ corresponds to the permutation $(1,2,3,4)$ of operations, and solution $\mathfrak{s}_6$ corresponds to permutation $(3,4,1,2)$. Both solutions are disproportionally favored by the way in which solutions are constructed and pheromone values are updated. The corresponding permutations are characterized by the fact that the operations of each job form a continuous block (or sequence), i.e., operation 4 is the immediate successor of operation 3, and operation 2 is the immediate successor of operation 1. Therefore, for this small problem instance, it holds that those solutions that correspond to permutations in which the operations of each job form a sequence are favored. In order to study if this holds in general, we introduce a measure



(a) Instance $patho\_1$, performance.

(b) Instance $patho\_1$, sequencing factors.

Fig. 6. (a) Shows the evolution of the average iteration quality obtained by AS_JSS_suc when applied to problem instances $patho\_1$ for 2000 iterations with $n_a = 10$ and with the evaporation rate ρ optimally chosen (in an experimental way). All the pheromone values were initially set to 0.5. The results are averaged over 100 runs (the error bars that show the standard deviation are shown every 50th iteration). (b) Shows the evolution of the average sequencing factors of the iteration-best solutions corresponding to the experiment that is shown in (a).

which we, henceforth, refer to as the *sequencing factor*. Given a permutation π that corresponds to a solution $\mathfrak{s}$, the sequencing factor of π is given by

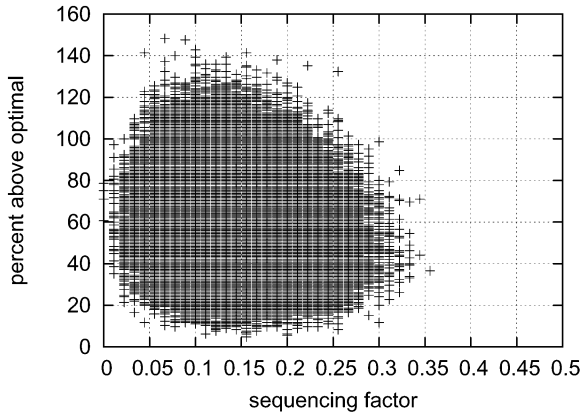
$$f_{\text{seq}}(\pi) \leftarrow \frac{\sum_{i=1}^{n-1} \delta(\pi, i, i+1)}{\sum_{\mathcal{J}_j \in \mathcal{J}} (|\mathcal{J}_j| - 1)} \tag{18}$$

where $\delta(\pi, i, i+1) = 1$ if the operations on positions i and $i+1$ of permutation π are from a same job, and $\delta(\pi, i, i+1) = 0$, otherwise. The minimum value of $f_{\text{seq}}(\pi)$ is 0, in case no two neighboring operations in π are from the same job. The maximum value of $f_{\text{seq}}(\pi)$ is 1, in case the jobs are in sequences in π. Corresponding to the experiments that are shown in Fig. 5(a) and (c), we plotted the sequencing factors of the iteration-best solutions over time in Fig. 5(b) and (d). It is interesting to observe that the sequencing factors of algorithm versions AS_JSS_suc and IB_JSS_suc for both instances strongly increase right from the start of the algorithm run. In case of AS_JSS_suc, they reach for both instances the maximum value of 1.0 and stay at this level until the algorithm is stopped. In case of IB_JSS_suc, the increase is delayed and the sequencing factors stop shortly before reaching 1.0. The use of local search strongly reduces the above described effects. The time (in terms of the iteration number) at which the sequencing factor reaches a stable level coincides with the point at which the evolution of the average iteration quality takes a sharp turn from decreasing average iteration quality to increasing average iteration quality. This indicates that the sequencing factor is correlated with the bias that is introduced by the fact that the systems are not competition-balanced. The sequencing factor can therefore be regarded as an indicator of this bias, which leads the algorithm at the beginning of the search process to a certain area of the search space that is characterized by the fact that the solutions in this area have a high sequencing factor. Once the algorithm has reached this area, this bias reduces and the selection pressure[9]—now being the strongest force—leads the algorithm to find the good solutions within this area.

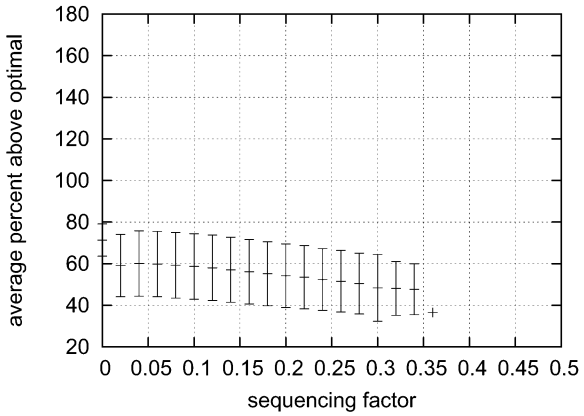
Our observations concerning the sequencing factors indicate an important fact. The bias introduced by the combination of an ACO algorithm with a problem instance that is not competition-balanced is not necessarily harmful. If, for example, the good solutions of a problem instance were characterized by a high sequencing factor, algorithms AS_JSS_suc and IB_JSS_suc should work quite well, and we should not observe noticeable second-order deception effects. In order to empirically support this claim, we generated such a problem instance, henceforth, denoted by $patho\_1$. This problem instance consists of 100 operations (i.e., ten jobs and ten machines). The first operation of each job has to be processed on machine $\mathcal{M}_1$, the second operation of each job has to be processed on machine $\mathcal{M}_2$, and so on. Furthermore, the processing times of the operations are as follows: $p(o) = 110 - (10 * j)$, $\forall\ o \in \mathcal{M}_j$, for $j = 1, \ldots, 10$. The solutions with maximal sequencing factor are optimal solutions to this problem instance. The results of applying AS_JSS_suc to instance $patho\_1$ are shown in Fig. 6. As expected AS_JSS_suc performs—except for the first about 100 iterations, where effects of random initial search can be observed—very well on this problem instance.

The characteristics of instance $patho\_1$ are, however, quite unusual. We studied characteristics of: 1) instance $patho\_1$; 2) well-known benchmark instances; and 3) randomly generated instances. For each problem instance, we randomly generated 10^6 solutions (as permutations of all operations), and created two plots from the results. The first plot is a scatter plot in which each data point corresponds to one of the constructed solutions. The sequencing factor of a solution is on the x axis and the solution quality (in terms of "percent above optimal") is on the y axis. The second plot presents the same information in a different way. For the second plot, we divided the range of the sequencing factors into intervals of equal size and categorized

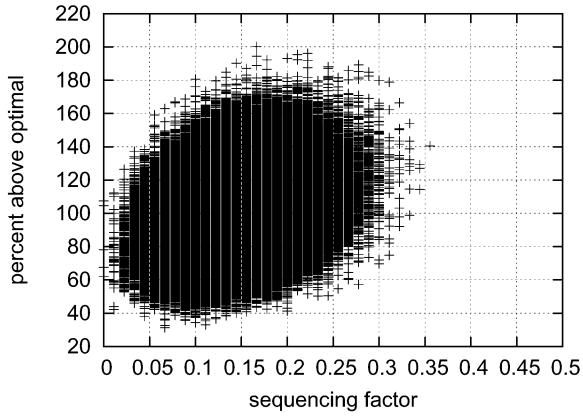
[9]We call the influence of solutions on the pheromone update depending on their quality *selection pressure*.



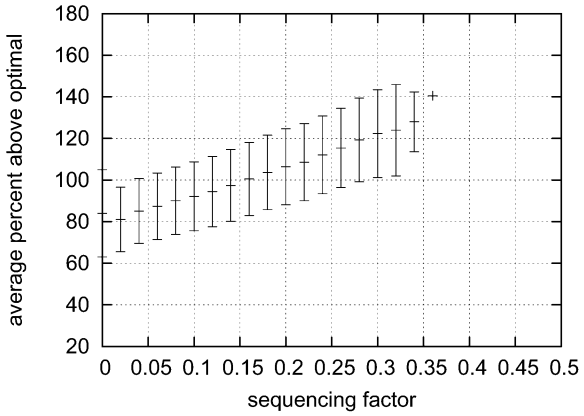
(a) Instance $patho\_1$.



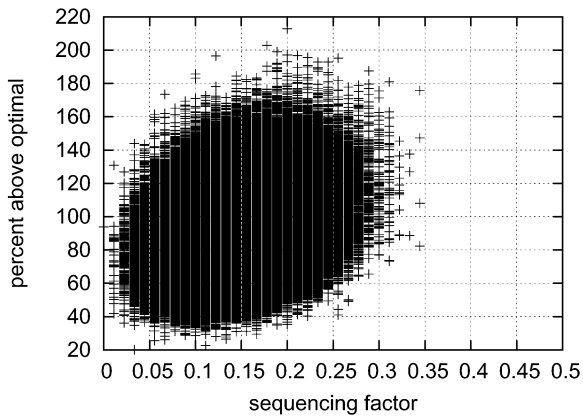
(b) Instance $patho\_1$.



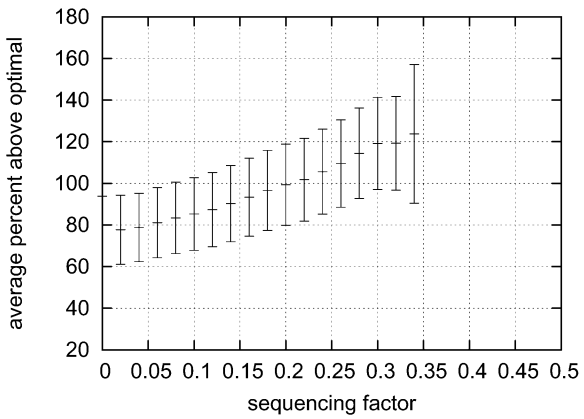
(c) Instance $ft10$.



(d) Instance $ft10$.



(e) Instance $10\times10\_1$.



(f) Instance $10\times10\_1$.

Fig. 7. Sequencing factor versus solution quality for JSS problem instances. Plots (a), (c), and (e) are scatter plots showing the sequencing factor and the solution quality of each randomly generated solution, whereas plots (b), (d), and (f) show the average quality (with standard deviation) of randomly generated solutions from ranges of sequencing factors.

the constructed solutions with respect to the interval they fall into. Then, for each sequencing factor interval, we plotted the average solution quality that was obtained together with error bars that show the standard deviation. Fig. 7 shows these two plots for three different instances: $patho\_1$, $ft10$, and a randomly generated instance $10\times10\_1$ that consists of 100 operations (i.e., ten jobs and ten machines). The characteristics of $ft10$ are typical of existing benchmark instances, while those of $10\times10\_1$ are typical of randomly generated instances. We observe that for both instance types (i.e., "existing benchmark" and "randomly generated") with increasing sequencing factor the average solution quality (of randomly generated solutions) degrades.[10] As can be seen in Fig. 7(b), the opposite is true for instance $patho\_1$.

To summarize, if a combination of an ACO algorithm and a problem instance is not a CBS, the search process may be subject to a bias. This bias may be harmful, resulting in second-

[10]Note that in Fig. 7, the sequencing factors of solutions do not exceed 0.4, because it is highly improbable to randomly generate solutions with sequencing factors higher than 0.4.

order deception effects, as in our ACO algorithms for the JSS problem. However, this bias is not necessarily harmful, as it may bias the search toward areas in the search space that contain good solutions. This was shown on the example of the pathological instance $patho\_1$.

V. Avoiding Second-Order Deception: Choice of an Appropriate Pheromone Model

In general, there may be several possible ways of avoiding second-order deception. In this paper, we focus on avoiding second-order deception by choosing an appropriate problem model of the CO problem under consideration in order to obtain an appropriate pheromone model. In this context, the notion of an *appropriate pheromone model* informally denotes a pheromone model that avoids second-order deception when coupled to the given construction mechanism and pheromone update. The comparison of different pheromone models has not received much attention in ACO research so far. One of the rare examples is the work by Roli *et al.* [36] on MAX-SAT. In fact, the common practice when applying ACO to a CO problem to which it has not been applied before is to choose the most natural problem model[11] and to derive the pheromone model from it. Then, either the chosen ACO algorithm works well (i.e., it provides competitive results), or, in case the algorithm shows a rather low performance, the reasons for the lack of success remain unclear and unexplored. This might be due to the use of additional algorithmic components, such as, for example, the local search that is used to improve solutions, which make it difficult to detect second-order deception. Therefore, the CO problem model—respectively, the pheromone model—is rarely identified as the cause of a bad algorithm performance.

The ideal solution would be to develop a model of the JSS problem such that the ACO algorithm outlined before leads to a CBS, independently of the JSS instance it is applied to. However, we were not able to find such a model of the JSS problem. Therefore, we decided to investigate two alternative JSS problem models that already exist.

A. Problem Model $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{pos}}$

Problem model $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{pos}}$ was introduced by Merkle and Middendorf in [27] for a scheduling problem related to the JSS problem. In this model, we are given a decision variable X_i for each position $i \in \{1, \ldots, n\}$ of a permutation of length n. The domain of a decision variable X_i is $D_i = \{1, \ldots, n\}$. Setting $X_i = j \in D_i$ corresponds to placing operation o_j on position i of the permutation being built. The pheromone model is derived as follows from this problem model. For each combination of a decision variable X_i and a domain value $j \in D_i$, we have a solution component $\mathfrak{c}_i^j$, and for each solution component, we have a pheromone trail parameter $\mathcal{T}_i^j$.

Using the mechanism of list scheduler algorithms, the solution construction process is as follows. Let i_t denote the index of the decision variable that has to be assigned a value in the current construction step $t \in \{1, \ldots, n\}$. The solution construction starts with an empty partial solution $\mathfrak{s}^p = \langle\rangle$ and with $i_t = 1$. Then, a solution component $\mathfrak{c}_{i_t}^j \in \mathfrak{N}(\mathfrak{s}^p)$ is added to the current partial solution $\mathfrak{s}^p$, where

$$\mathfrak{N}(\mathfrak{s}^p) = \left\{ \mathfrak{c}_{i_t}^k \mid o_k \in \mathcal{O}_t \right\}. \tag{19}$$

When adding the solution component $\mathfrak{c}_{i_t}^j$ to $\mathfrak{s}^p$, we also set $i_t \leftarrow i_t + 1$. This means that we fill permutations from left to right.

Merkle and Middendorf proposed two different ways of defining the transition probabilities. In the so-called *standard* way, the transition probabilities are defined as follows:

$$\mathrm{p}\left(\mathfrak{c}_{i_t}^j \mid \mathcal{T}\right) = \frac{\tau_{i_t}^j}{\sum_{\mathfrak{c}_{i_t}^k \in \mathfrak{N}(\mathfrak{s}^p)} \tau_{i_t}^k} \qquad \forall\, \mathfrak{c}_{i_t}^j \in \mathfrak{N}(\mathfrak{s}^p). \tag{20}$$

However, the problem with this way of defining the transition probabilities is the following. Imagine a situation in which τ_i^j, which represents the desirability to place operation o_j on position i, is quite high and all pheromone values τ_k^j, with $k \in \{i+1, \ldots, n\}$, are quite low. If for stochastic reasons operation o_j is not placed at position i, then the probability to place it at any of the immediately following positions is quite low. This means that there is a high probability that this operation will be placed much later in the permutation, which often results in low quality solutions. To favor the placement of operation o_j, if not at position i, at least not too far away from position i, the so-called *summation evaluation rule* for defining the transition probabilities was proposed by Merkle and Middendorf in [27]

$$\mathrm{p}\left(\mathfrak{c}_{i_t}^j \mid \mathcal{T}\right) = \frac{\sum_{l=1}^{i_t} \tau_l^j}{\sum_{\mathfrak{c}_{i_t}^k \in \mathfrak{N}(\mathfrak{s}^p)} \sum_{l=1}^{i_t} \tau_l^k} \qquad \forall\, \mathfrak{c}_{i_t}^j \in \mathfrak{N}(\mathfrak{s}^p). \tag{21}$$

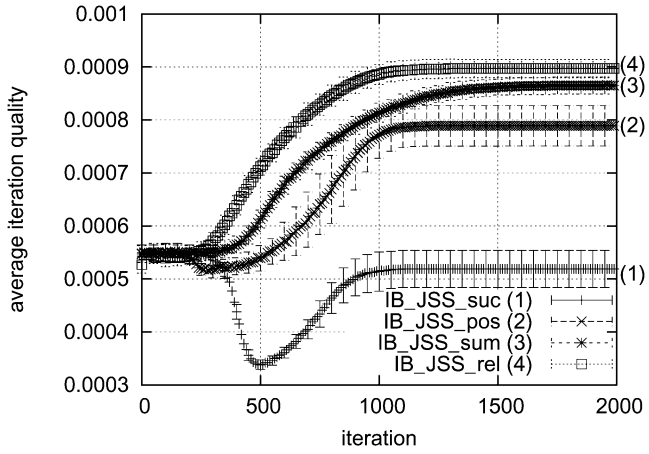
With this definition of the transition probabilities, the probability to assign an operation to the current permutation position is proportional to the sum of the pheromone values for assigning this operation to any of the already assigned permutation positions.

The difference between models $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{pos}}$ and $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{suc}}$ is the following. With the pheromone model derived from $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{suc}}$, successor relationships are learned between operations; that is, permutations are built from left to right by making the choice of the operation for the current position depending on the operation that was chosen for the previous position. On the contrary, when using the pheromone model derived from $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{pos}}$, learning concerns where to position operations in the permutation that is built.

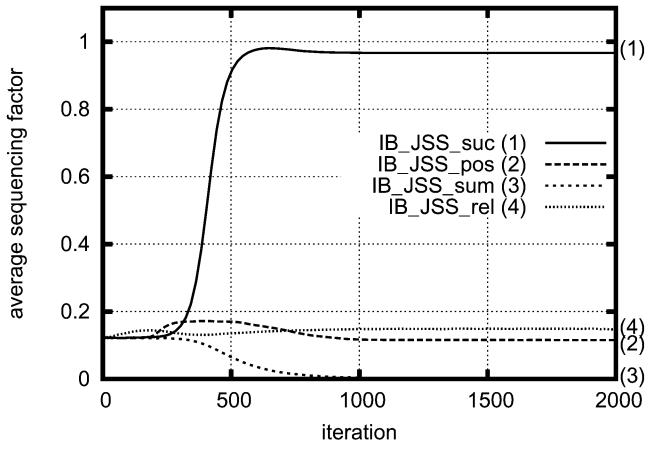
B. Problem Model $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{rel}}$

Both model $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{suc}}$ and model $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{pos}}$ may be considered as unnatural models of the JSS problem. To explain what we mean, we introduce the notion of *related* operations. Henceforth, we call two operations o_i and o_j related, if they have to be processed on a same machine. We denote the set of operations that are related to an operation o_i with $\mathcal{R}_i$. A solution to a JSS instance basically consists of a processing order for each pair of related operations. However, with model $\mathcal{P}_{\mathrm{JSS}}^{\mathrm{suc}}$ successor relationships between operations that are possibly not related are learned. This is not necessary and therefore quite unnatural. The position

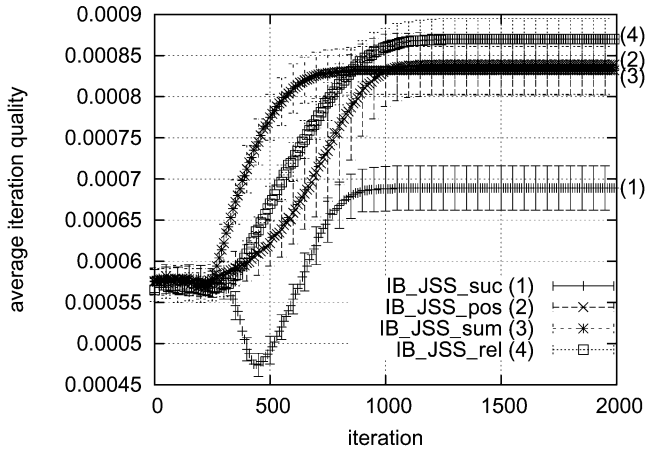
[11] This often corresponds to the problem model as it can be found in the literature.



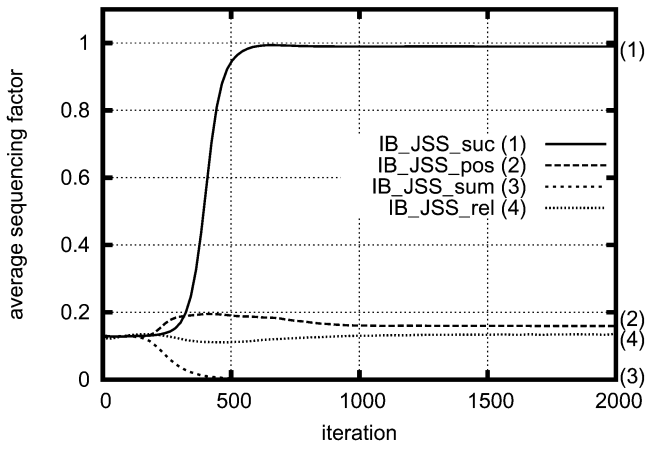
(a) Instance $ft10$, IB-update, performance.



(b) Instance $ft10$, IB-update, sequencing factors.



(c) Instance $orb08$, IB-update, performance.



(d) Instance $orb08$, IB-update, sequencing factors.

Fig. 8. (a) and (c) Shows the comparison between algorithm versions IB_JSS_suc, IB_JSS_pos, IB_JSS_sum, and IB_JSS_rel when applied to problem instances $ft10$ and $orb08$ for 2000 iterations with $n_a = 10$ and with the evaporation rate ρ optimally chosen (in an experimental way) for each algorithm version. All the pheromone values were initially set to 0.5. The results are averaged over 100 runs (the error bars that show the standard deviation are shown every 50th iteration). (b) and (d) Shows the evolution of the sequencing factors of the iteration-best solutions corresponding to the experiments that are shown in (a) and (c).

learning with model $\mathcal{P}_{\text{JSS}}^{\text{pos}}$ is likewise unnatural, because the relevant information for a solution is the processing order between each pair of related operations. Based on these considerations, Blum and Sampels in [6] proposed problem model $\mathcal{P}_{\text{JSS}}^{\text{rel}}$, which is outlined in the following.

Model $\mathcal{P}_{\text{JSS}}^{\text{rel}}$ consists of a binary decision variable for each pair of operations $o_s, o_r \in \mathcal{O}(o_s \neq o_r)$ that are related. If $s < r$, this variable is denoted by X_{sr}, by X_{rs}, otherwise. Setting a decision variable X_{ij} to 1 means that o_i has to be processed before o_j, whereas setting X_{ij} to 0 means that o_j has to be processed before o_i. Note that this model contains redundancy. Consider, for example, a machine with three operations o_1, o_2 and o_3. Variable assignments $X_{1,2} = 1$ and $X_{2,3} = 1$ imply $X_{1,3} = 1$. In this model, we have for each binary decision variable X_{ij} two solution components: $\mathfrak{c}_{ij}^1$ corresponding to $X_{ij} = 1$, and $\mathfrak{c}_{ij}^0$ corresponding to $X_{ij} = 0$. Associated with each solution component $\mathfrak{c}_{ij}^x$ (where $x \in \{0, 1\}$), we have a pheromone trail parameter $\mathcal{T}_{ij}^x$ with value τ_{ij}^x.

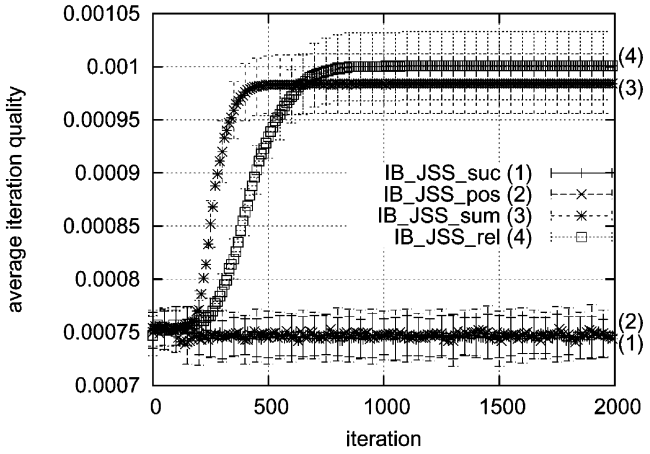
For constructing solutions, we again use the mechanism of list scheduler algorithms. However, the sequence of *groups* of solution components that is built—as we will explain below—does not directly correspond to the permutation of operations that is built in parallel by the list scheduler algorithm. The solution construction starts with an empty partial solution $\mathfrak{s}^p = \langle\rangle$. Then, at each construction step it is first checked if an operation $o_k \in \mathcal{O}_t$ exists such that $\mathcal{R}_k \cap \mathcal{O}^+ = \emptyset$; that is, if an operation o_k among the ones that can be scheduled exists for which no related operation remains unscheduled. If this is the case, operation o_k is regarded as scheduled (i.e., it is removed from $\mathcal{O}_t$) and the algorithm proceeds to the next iteration.[12] This means that $\mathfrak{s}^p$ remains the same, whereas the permutation of operations built in parallel by the list scheduler algorithm is changed by appending operation o_k. If no such operation o_k exists, a group of solution components $\mathfrak{g}_j \in \mathfrak{N}(\mathfrak{s}^p)$ is added to the current partial solution $\mathfrak{s}^p$, where

$$\mathfrak{N}(\mathfrak{s}^p) \leftarrow \{\mathfrak{g}_i \mid o_i \in \mathcal{O}_t\} \tag{22}$$

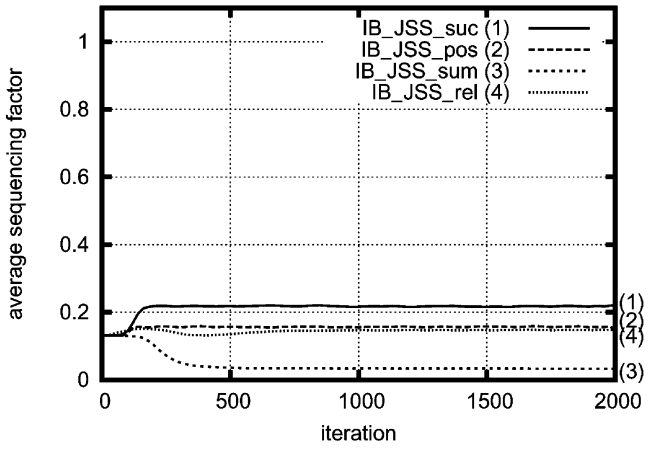
with

$$\mathfrak{g}_i \leftarrow \left\{\mathfrak{c}_{ij}^1 \mid o_j \in \mathcal{R}_i \cap \mathcal{O}^+, i < j\right\} \cup \left\{\mathfrak{c}_{li}^0 \mid o_l \in \mathcal{R}_i \cap \mathcal{O}^+, l < i\right\}. \tag{23}$$

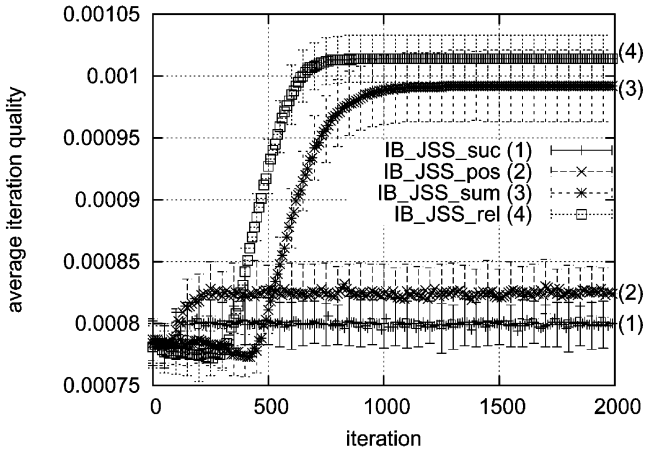
[12]Note that this can be done because such an operation does not compete with any of the remaining unscheduled operations, that is, the values of all decision variables concerning this operation are already specified.



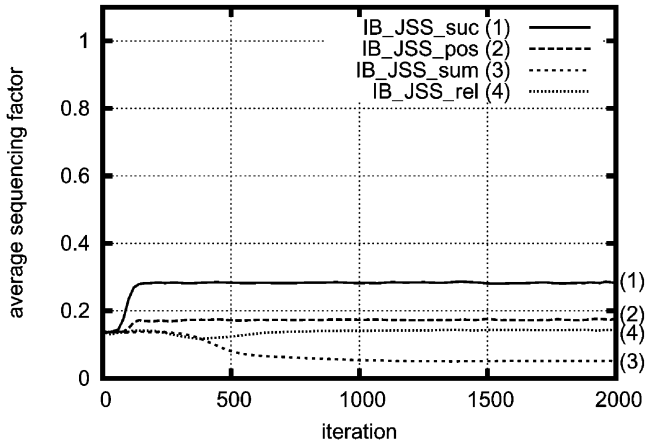
(a) Instance $ft10$, IB-update + local search, performance.



(b) Instance $ft10$, IB-update + local search, sequencing factors.



(c) Instance $orb08$, IB-update + local search, performance.



(d) Instance $orb08$, IB-update + local search, sequencing factors.

Fig. 9. (a) and (c) Shows the comparison between algorithm versions IB_JSS_suc, IB_JSS_pos, IB_JSS_sum, and IB_JSS_rel (all using local search) when applied to problem instances $ft10$ and $orb08$ for 2000 iterations with $n_a = 10$ and with the evaporation rate ρ optimally chosen (in an experimental way) for each algorithm version. All the pheromone values were initially set to 0.5. The results are averaged over 100 runs (the error bars that show the standard deviation are shown every 50th iteration). (b) and (d) Shows the evolution of the sequencing factors of the iteration-best solutions corresponding to the experiments that are shown in (a) and (c).

The solution components of a group $\mathfrak{g}_i$ determine the setting of the decision variables involving operation o_i and operations that are related to o_i and that are unscheduled. In [6], the transition probabilities were defined as follows:

$$\mathbf{p}(\mathfrak{g}_i \mid \mathfrak{N}(\mathfrak{s}^p)) = \frac{\min\left\{\min_{\mathfrak{c}^1_{ij}\in\mathfrak{g}_i} \tau^1_{ij}, \min_{\mathfrak{c}^0_{li}\in\mathfrak{g}_i} \tau^0_{li}\right\}}{\sum_{\mathfrak{g}_k\in\mathfrak{N}(\mathfrak{s}^p)} \min\left\{\min_{\mathfrak{c}^1_{kj}\in\mathfrak{g}_k} \tau^1_{kj}, \min_{\mathfrak{c}^0_{lk}\in\mathfrak{g}_k} \tau^0_{lk}\right\}} \quad (24)$$

$\forall\ \mathfrak{g}_i \in \mathfrak{N}(\mathfrak{s}^p)$. In this way, the probability for each $\mathfrak{g}_i$ is proportional to the minimum of the values of the pheromone trail parameters on the solution components that constitute this group. This is a reasonable choice, because if this minimum is low it means that there is at least one operation left, which is unscheduled and related to operation o_i, that should be scheduled before o_i.

C. Results

We tested four ACO algorithm versions using the IB-update rule. These versions are based on the three pheromone models derived from problem models $\mathcal{P}^{\text{suc}}_{\text{JSS}}$, $\mathcal{P}^{\text{pos}}_{\text{JSS}}$, and $\mathcal{P}^{\text{rel}}_{\text{JSS}}$. They are denoted by IB_JSS_suc (corresponding to problem model $\mathcal{P}^{\text{suc}}_{\text{JSS}}$), IB_JSS_pos (corresponding to problem model $\mathcal{P}^{\text{pos}}_{\text{JSS}}$ with standard pheromone evaluation), IB_JSS_sum (corresponding to problem model $\mathcal{P}^{\text{pos}}_{\text{JSS}}$ and summation evaluation rule), and IB_JSS_rel (corresponding to problem model $\mathcal{P}^{\text{rel}}_{\text{JSS}}$). All algorithm versions were applied—with and without the use of local search—to problem instances $ft10$ and $orb08$.

The results are shown in Figs. 8 (without local search) and 9 (with local search). First of all, we observe—in particular, when local search is used—a clear advantage of the algorithm versions IB_JSS_rel and IB_JSS_sum over the other two. When local search is not used, algorithm version IB_JSS_pos outperforms algorithm version IB_JSS_suc. When local search is used, these two algorithm versions have a quite low performance. We also observe that the summation evaluation rule clearly improves on the standard way of defining the transition probabilities. Finally, concerning the comparison of IB_JSS_rel with IB_JSS_sum, we note, generally, a slight advantage of IB_JSS_rel.

Concerning algorithm versions IB_JSS_pos, IB_JSS_sum, and IB_JSS_rel, we only observe

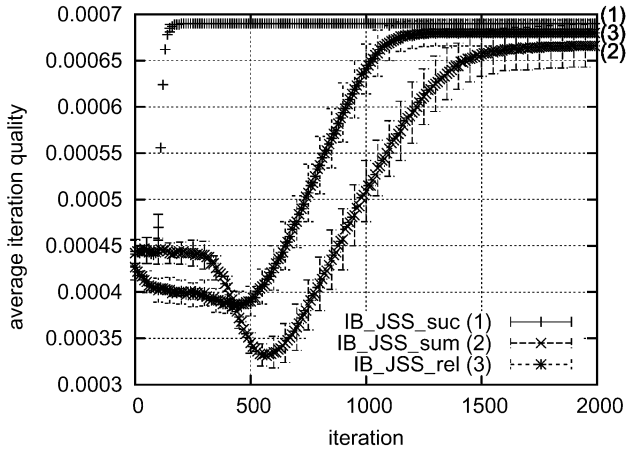


Fig. 10. Comparison of IB_JSS_suc, IB_JSS_sum, and IB_JSS_rel applied to problem instance $patho\_1$ for 2000 iterations with $n_a = 10$ and with the evaporation rate ρ optimally chosen (in an experimental way) for each algorithm version. All the pheromone values were initially set to 0.5. The results are averaged over 100 runs (the error bars that show the standard deviation are shown every 50th iteration).

slight second-order deception effects. IB_JSS_rel seems to be the least affected by any harmful bias. In this context, it is interesting to study the evolution of the average sequencing factors. As mentioned before, the average sequencing factors of the results obtained by IB_JSS_suc without the use of local search steadily increase until some stable level is reached close to 1. When local search is added, a stable level is already reached at about 0.2. In contrast, the average sequencing factors of the results obtained by IB_JSS_pos only slightly increase before they start to decrease again. Interestingly, the average sequencing factors belonging to IB_JSS_sum decrease strongly from the start until a stable level is reached at 0. This indicates that the use of the summation evaluation rule for generating the transition probabilities [see (21)] introduces a strong bias toward solutions with very low sequencing factors. Accordingly, we would expect this algorithm version to have difficulties when applied to a problem instance such as $patho\_1$ (see Section IV-B). This is confirmed by the results that are shown in Fig. 10. Finally, the average sequencing factors belonging to IB_JSS_rel stay at approximately the same level (i.e., around 0.13) during the whole run, which indicates that there is no bias that is correlated to the sequencing factor.[13]

In summary, we can say that even though, in general, combinations of IB_JSS_rel and JSS problem instances are not CBSs (see for example the combination with instance jss_simple_inst), the algorithm does not seem to suffer from a harmful form of bias that causes overly strong second-order deception effects. Algorithm IB_JSS_sum on the other hand, is subject to a strong bias toward solutions with low sequencing factors. However, as those solutions in nonpathological problem instances are generally good solutions, the bias is generally not harmful. Concerning algorithm IB_JSS_pos, we observed that it is a bad match with the local search that we used to improve solutions, i.e., IB_JSS_pos did not seem to be able to improve over time when local search was used.

[13]Note that the sequencing factors obtained by IB_JSS_rel might be slightly biased by the fact that operations for which no unscheduled and related operations exist are automatically scheduled.

VI. Conclusion and Future Work

In this paper, we have studied the occurrence of harmful bias in the search process of ACO algorithms. The visible effects of harmful bias, i.e., the decrease in algorithm performance over time, are labeled *second-order deception effects*. We have introduced the concept of *CBSs* defined over combinations of ACO algorithms with problem instances. Competition-balanced systems provide a way of distinguishing between fair and unfair competitions between solution components, i.e., the components of which solutions are composed. With the ant system algorithm for the ATSP, we have shown an example of a fair competition. In this example, we did not observe second-order deception effects, which led us to conjecture the nonexistence of any bias. On the other hand, we have shown on the example of JSS that combinations of ACO algorithms with problem instances that are not competition-balanced may suffer from a harmful bias that causes a degeneration of algorithm performance over time, i.e., second-order deception effects. As a way of avoiding second-order deception effects, we have proposed the careful choice of an appropriate pheromone model.

Future work will consider other ways of avoiding second-order deception effects, such as the use of different types of pheromone update rules. We also aim at extending our work to other combinatorial optimization problems. In the case of the JSS problem, we discovered that certain pheromone models bias the search process toward solutions with certain sequence properties. Unfortunately, this finding cannot be generalized to other optimization problems. Therefore, our future work aims at more general statements that apply to groups of combinatorial optimization problems, for example, subset problems or permutation problems. Finally, we aim to prove theoretically that if the combination of an ACO algorithm with a problem instance is a CBS, it does not suffer from second-order deception.

References

[1] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA J. Comput.*, vol. 3, pp. 149–156, 1991.

[2] J. Blażewicz, W. Domschke, and E. Pesch, "The job shop scheduling problem: Conventional and new solution techniques," *Eur. J. Oper. Res.*, vol. 93, pp. 1–33, 1996.

[3] C. Blum, "Theoretical and practical aspects of ant colony optimization," Ph.D. dissertation, Akademische Verlagsgesellschaft Aka GmbH, Berlin, Germany, 2004. Vol. 282, Dissertationen zur Künstlichen Intelligenz.

[4] C. Blum and M. Dorigo, "Deception in ant colony optimization," in *Lecture Notes in Computer Science*, M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, Eds. Berlin, Germany, 2004, vol. 3172, Proc. 4th Int. Workshop Ant Colony Opt. Swarm Intell. (ANTS), pp. 119–130.

[5] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[6] C. Blum and M. Sampels, "Ant colony optimization for FOP shop scheduling: A case study on different pheromone representations," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 2, Los Alamitos, CA, 2002, pp. 1558–1563.

[7] ——, "When model bias is stronger than selection pressure," in *Lecture Notes in Computer Science*, J. J. Merelo Guervós *et al.*, Eds. Berlin, Germany, 2002, vol. 2439, Proc. 7th Int. Conf. Parallel Prob. Solving From Nature (PPSN-VII), pp. 893–902.

[8] C. Blum, M. Sampels, and M. Zlochin *et al.*, "On a particularity in model-based search," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, W. B. Langdon *et al.*, Eds., San Francisco, CA, 2002, pp. 35–42.

[9] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian, "Ant system for job-shop scheduling," *Belgian J. Oper. Res. Statis. Comput. Sci. (JORBEL)*, vol. 34, no. 1, pp. 39–53, 1994.
[10] C. Cotta and J. M. Troya, "Genetic forma recombination in permutation flowshop problems," *Evol. Comput.*, vol. 6, no. 1, pp. 25–44, 1998.
[11] K. Deb and D. E. Goldberg, "Analyzing deception in trap functions," in *Foundations of Genetic Algorithms 2*, L. D. Whitley, Ed. San Mateo, CA: Morgan Kaufmann, 1993, pp. 93–108.
[12] J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels, "The self-organizing exploratory pattern of the argentine ant," *J. Insect Behavior*, vol. 3, pp. 159–168, 1990.
[13] M. Dorigo, "Optimization, Learning and Natural Algorithms," Ph.D. dissertation (in Italian), Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.
[14] M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, Eds., *Lecture Notes in Computer Science.* Berlin, Germany, 2004, vol. 3172, Proc. 4th Int. Workshop Ant Colony Opt. Swarm Intell. (ANTS).
[15] M. Dorigo, L. M. Gambardella, M. Middendorf, and T. Stützle, "Special section on ant colony optimization," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 317–365, Aug. 2002.
[16] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive Feedback as a Search Strategy," Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, Tech. Rep. 91-016, 1991.
[17] ——, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern.—Part B*, vol. 26, no. 1, pp. 29–41, Feb. 1996.
[18] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.
[19] L. J. Eshelmann, R. A. Caruana, and J. D. Schaffer, "Biases in the crossover landscape," in *Proc. 3rd Int. Conf. Genetic Algorithms (ICGA)*, 1989, pp. 10–19.
[20] B. Giffler and G. L. Thompson, "Algorithms for solving production scheduling problems," *Oper. Res.*, vol. 8, pp. 487–503, 1960.
[21] F. Glover and G. Kochenberger, Eds., *Handbook of Metaheuristics*. Norwell, MA: Kluwer, 2002.
[22] D. E. Goldberg, "Simple genetic algorithms and the minimal deceptive problem," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. London, U.K.: Pitman, 1987, pp. 74–88.
[23] R. A. Holmgren, *A First Course in Discrete Dynamical Systems*. Berlin, Germany: Springer-Verlag, 1996.
[24] C. Igel and P. Stagge, "Effects of phenotypic redundancy in structure optimization," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 74–85, 2002.
[25] E. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, *The Traveling Salesman Problem*. New York: Wiley, 1985.
[26] N. F. McPhee, R. Poli, and J. E. Rowe, "A schema theory analysis of mutation size biases in genetic programming with linear representations," in *Proc. Congr. Evol. Comput. (CEC)*, 2001, pp. 1078–1085.
[27] D. Merkle and M. Middendorf, "An ant algorithm with a new pheromone evaluation rule for total tardiness problems," in *Lecture Notes in Computer Science.* Berlin, Germany, 2000, vol. 1803, Proc. EvoWorkshops 2000, pp. 287–296.
[28] ——, "Modeling ACO: Composed permutation problems," in *Lecture Notes in Computer Science*, M. Dorigo, G. Di Caro, and M. Sampels, Eds. Berlin, Germany, 2002, vol. 2463, Proc. 3rd Int. Workshop Ant Algorithms, Ant Colonies to Artif. Ants (ANTS), pp. 149–162.
[29] ——, "Modeling the dynamics of ant colony optimization algorithms," *Evol. Comput.*, vol. 10, no. 3, pp. 235–262, 2002.
[30] J. Montgomery, M. Randall, and T. Hendtlass, "Search bias in constructive metaheuristics and implications for ant colony optimization," in *Lecture Notes in Computer Science*, M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, Eds. Berlin, Germany, 2004, vol. 3172, Proc. 4th Int. Workshop, Ant Colony Opt. Swarm Intell. (ANTS), pp. 391–398.
[31] J. F. Muth and G. L. Thompson, *Industrial Scheduling*. Englewood Cliffs, NJ: Prentice-Hall, 1963.
[32] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job-shop problem," *Manage. Sci.*, vol. 42, no. 2, pp. 797–813, 1996.
[33] C. C. Palmer and A. Kershenbaum, "Representing trees in genetic algorithms," in *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. Bristol, U.K.: Inst. of Physics, Oxford Univ. Press, 1997, pp. G1.3:1–8.
[34] R. Poli and N. F. McPhee *et al.*, "Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size," in *Lecture Notes in Computer Science*, J. F. Miller *et al.*, Eds. Berlin, Germany, 2001, vol. 2038, Proc. 4th Eur. Conf. Genetic Program. (EuroGP), pp. 126–142.
[35] R. Poli, C. R. Stephens, A. H. Wright, and J. E. Rowe, "On the search biases of homologuous crossover in linear genetic programming and variable-length genetic algorithms," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, W. B. Langdon, Ed., San Francisco, CA, 2002, pp. 868–876.
[36] A. Roli, C. Blum, and M. Dorigo, "ACO for maximal constraint satisfaction problems," in *Proc. 4th Metaheuristics Int. Conf. (MIC)*, vol. 1, Porto, Portugal, 2001, pp. 187–191.
[37] F. Rothlauf and D. E. Goldberg, "Prüfer numbers and genetic algorithms: A lesson on how the low locality of an encoding can harm the performance of GAs," in *Lecture Notes in Computer Science.* Berlin, Germany, 2000, vol. 1917, Proc. 6th Int. Conf. Parallel Prob. Solving From Nature (PPSN-VI), pp. 395–404.
[38] K. Vekaria and C. Clack, "Biases introduced by adaptive recombination operators," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, vol. 1, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., San Fransisco, CA, 1999, pp. 670–677.
[39] D. L. Whitley and S. B. Rana, "Representation, search, and genetic algorithms," in *Proc. 14th Nat. Conf. Artif. Intell. (AAAI)*, Menlo Park, CA, 1997, pp. 497–502.

Christian Blum received the M.S. degree in mathematics from the Universität Kaiserslautern, Kaiserslautern, Germany, in 1998 and the Ph.D. degree in applied sciences from the Université Libre de Bruxelles (ULB), Brussels, Belgium, in 2004.

From 1999 to 2000, he was with the Advanced Computation Laboratory (ACL), Imperial Cancer Research Fund (ICRF), London, U.K., and from 2000 to 2004, he was with IRIDIA, ULB. He currently holds a Postdoctoral Fellowship at the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. His research interests include metaheuristics in combinatorial optimization with a focus on theoretical and practical aspects of ant colony optimization. Current subject of his research is the hybridization of ant colony optimization with more classical artificial intelligence and operations research methods. In 2004, he coorganized the First International Workshop on Hybrid Metaheuristics (HM 2004), and was a Coeditor of the Proceedings of ANTS 2004, Fourth International Workshop on Ant Algorithms and Swarm Intelligence.

Marco Dorigo (S'92–M'93–SM'96) received the Laurea (Master of Technology) degree in industrial technologies engineering and the Ph.D. degree in information and systems electronic engineering from Politecnico di Milano, Milan, Italy, in 1986 and 1992, respectively, and the title of Agrégé de l'Enseignement Supérieur from the Université Libre de Bruxelles (ULB), Brussels, Belgium, in 1995.

From 1992 to 1993, he was a Research Fellow at the International Computer Science Institute, Berkeley, CA. In 1993, he was a NATO-CNR Fellow, and from 1994 to 1996 a Marie Curie Fellow. Since 1996, he has been a Tenured Researcher of FNRS, Belgian National Fund for Scientific Research, and a Research Director at the Artificial Intelligence Laboratory, IRIDIA, ULB. He is the inventor of the ant colony optimization metaheuristic. His current research interests include metaheuristics for discrete optimization, swarm intelligence, and swarm robotics.

Dr. Dorigo was awarded the Italian Prize for Artificial Intelligence in 1996 and the Marie Curie Excellence Award in 2003. He is a member of the Editorial Boards of numerous international journals, including *Adaptive Behavior*, *AI Communications*, *Artificial Life*, *Evolutionary Computation*, *Journal of Heuristics*, *Cognitive Systems Research*, and the *Journal of Genetic Programming and Evolvable Machines*. He is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS.