Ant Colony Optimization for Path Planning in Search and Rescue Operations

Michael Morin[a,*], Irène Abi-Zeid[a], Claude-Guy Quimper[b]

[a]*Department of Operations and Decision Systems, Université Laval, Québec, Canada*
[b]*Department of Computer Science and Software Engineering, Université Laval, Québec, Canada*

Abstract

In search and rescue operations, an efficient search path, colloquially understood as a path maximizing the probability of finding survivors, is more than a path planning problem. Maximizing the objective adequately, i.e., quickly enough and with sufficient realism, can have substantial positive impact in terms of human lives saved. In this paper, we address the problem of efficiently optimizing search paths in the context of the NP-hard optimal search path problem with visibility, based on search theory. To that end, we evaluate and develop ant colony optimization algorithm variants where the goal is to maximize the probability of finding a moving search object with Markovian motion, given a finite time horizon and finite resources (scans) to allocate to visible regions. Our empirical results, based on evaluating 96 variants of the metaheuristic with standard components tailored to the problem and using realistic size search environments, provide valuable insights regarding the best algorithm configurations. Furthermore, our best variants compare favorably, especially on the larger and more realistic instances, with a standard greedy heuristic and a state-of-the-art mixed-integer linear program solver. With this research, we add to the empirical body of evidence on an ant colony optimization algorithms configuration and applications, and pave the way to the implementation of search path optimization in operational decision support systems for search and rescue.

Keywords: Evolutionary computations, Search and rescue, Optimal search path planning, Ant colony optimization, Humanitarian operations

*Corresponding author
Email address: `Michael.Morin@osd.ulaval.ca` (Michael Morin)

1. Introduction

The act of searching is an important part of many humanitarian operations, such as search and rescue (SAR), minesweeping and of many surveillance or homeland security operations for the purpose of protecting individuals, resources or infrastructures from current or future threats. Teams of searchers on the ground (humans and/or canine), in aircraft, in vessels as well as autonomous unmanned vehicles (robots) may search for survivors, land or underwater mines, or illicit activities and abnormal behaviors. In the event of natural disasters, such as earthquakes, floods, landslides, and other catastrophes involving collapsed buildings, aircraft, or maritime vessels, SAR operations must be quickly organized and deployed in order to locate and rescue survivors. This is also the case for smaller scale, albeit more frequent emergencies, such as persons who might have disappeared in water, a child who might have been lost, a confused person who might have wandered off, a hiker missing in the woods, etc. In Canada alone, there are thousands of air, maritime and ground SAR incidents every year (Minister of National Defence, 2013). Searching is, of course, also conducted by law enforcement agencies who wish to locate and neutralize threats.

But, how and where to search? The answer lies in efficient search planning that ensures the best use of scarce and constrained search resources. This implies defining search areas and/or search paths that maximize the chances of an operation's success, namely finding the search objects. Search planning is extremely complex since it is normally conducted under time pressure, in the presence of uncertain whereabouts, uncertain detectability, uncertain conditions of the search objects, and in degraded and rapidly changing conditions. The recent European migrant crisis and the human tragedies in the Mediterranean have emphasized the importance of efficient searches to quickly locate and rescue survivors.

In response to the first large-scale search operations, namely the hunt for enemy submarines off the Atlantic Coast during WWII, search theory was developed as one of the earlier subdisciplines of Operations Research, first classified and later published in the open literature (Charnes & Cooper, 1958). One of the problems addressed by search theory is the optimal search path (OSP) for a moving search object with uncertain location and detectability, an NP-hard problem (Trummel & Weisinger, 1986). In recent years, search theory has been used for planning of search paths of autonomous robots in structured environments or of unmanned aerial vehicles in large areas outdoors (Lau et al., 2008; Sato & Royset, 2010; Goerzen et al., 2010). Applications include humanitarian operations, mine countermeasures (Paull et al., 2012; Williams, 2010), evacuations following a disaster (Yuan & Wang, 2009), seabed surveys in harbors and waterways (Fang & Anstee, 2010), and search and rescue/recovery operations (Morin et al., 2009, 2010; Berger et al., 2012; Lo et al., 2012; Stone et al., 2016).

In the OSP formulation, it is generally assumed that a searcher can only scan its location. This, however, is an unrealistic assumption in an operational context, since regions other than the searcher's location may be visible from afar. To remedy this situation and adapt the problem representation to real-life, we formulated in Morin et al. (2009, 2010) the optimal searcher path problem with visibility (OSPV) and proposed algorithms based on ant colony optimization (ACO) a population-based, general stochastic local search technique (Hoos & Stützle, 2004; Dorigo & Blum, 2005). ACO have been applied to a wide area of problems and some of their recent successes in practical applications include Jovanovic et al. (2019) for the block relocation problem, Yu et al. (2019) for 3D path planning with dense obstacles, Verbeeck et al. (2014) on orienteering, i.e., the problem of selecting destinations and planning an optimal path to these selected locations, for humanitarian relief (Zhu et al., 2019), for disaster relief operations (Yi & Kumar, 2007), and for unmanned aerial vehicles path planning (Mirjalili et al., 2020).

In this paper, we generalize and improve our previous results by introducing the Ant Search Path with Visibility (ASPV) algorithm. We describe and discuss the outcomes of a thorough experimentation, conducted using 96 algorithmic variants, where a variant is a combination of pheromone initialization scheme, pheromone update scheme, diversification and intensification mechanisms. Based on realistic problem instances sizes, we compare the performance of our best ASPV algorithm variants with those obtained through a Mixed-Integer Linear Program (MILP) using the ILOG CPLEX solver as well as with a simple greedy heuristic. We show that our algorithms produce search paths with higher probabilities of success in shorter time. Our results provide an empirical contribution to the literature on the performance of ACO algorithms in general, and a first practical contribution towards the implementation of search pattern optimization in the Advanced Search Planning Tool, the Canadian decision support systems for SAR currently used in operations.

The rest of the paper is organized as follows. Section 2 provides an overview of related literature. Section 3 formally describes the OSPV problem. Section 4 presents the ASPV algorithm as well as pheromone boosting and restarts. Section 5 describes the experimental methodology. Section 6 contains the results along with a discussion. We conclude in Section 7.

2. Related literature

Problems of search theory may be formulated differently depending on the characteristics of the situation being addressed (Vermeulen & Van Den Brink, 2005; Stone, 2004), and on the measure of performance used, such as the probability of detection or the expected time to detection (Richardson, 2014). One of the main distinctions is whether the search object is moving or stationary. In two-sided search problems (also known as search games), the search object is active, i.e., its moves depend on the searcher's actions. It may be cooperative (e.g.*rendez-vous* search) or evading (e.g.*pursuit-evasion*). In a one-sided search problem, i.e., when the object's motion model does not depend on the searcher's actions, problems are again divided into two groups depending on the searcher's movement constraints: the optimal search path problems where the searcher is constrained to follow a path, also called path-constrained moving target search problem (Eagle & Yee, 1990), and the optimal search density problems where no such constraint is formulated (Lau et al., 2008). Two main types of motion models are considered: conditionally deterministic motion where the trajectory of a search object depends only on its initial position and Markovian motion models where an object's movement at a given point in time solely depends on its current location (Raap et al., 2017a). One type of search problem is *detection search*, where the search stops after the first detection (i.e., there is no target tracking as in surveillance search).

Search effort may be continuous or discrete. In the *continuous* case, effort may be allocated as finely as necessary over the entire search space (e.g., time spent by an aircraft over a set of regions) (Stewart, 1979). In this case, the objective function is convex and the constraints of the problem form a convex set. As for *discrete* search effort, it may be measured by the total number of searchers to deploy over an area of interest or by the total number of scans to allocate to a set of visible regions (Berger & Lo, 2015; Foraker et al., 2016; Raap et al., 2017b).

The optimal search path (OSP) is a single-sided detection search for a moving object with uncertain location and detectability. A solution to the OSP problem is a path on a graph that maximizes the probability of finding the object. This is different from classical path planning where the aim is often to plan a path from an initial point to a known destination. The OSP has been an active research topic since the introduction in 1979 of the first algorithm to solve it (Stewart, 1979). It is still attracting a lot of attention due to its applications in the robotics literature (Grogan et al., 2018; Raap et al., 2019). It is NP-hard problem (Trummel & Weisinger, 1986), and a lot of work to date has consisted of developing bounding techniques for the branch and bound (BB) algorithm presented by Stewart (1979) or of using a model and solve approaches with a bound (Simard et al., 2015) or without such a bound (Morin et al., 2012; Morin & Quimper, 2014). Another approach has been to use metaheuristics such as ACO (Ding & Pan, 2011; Perez-Carabaza et al., 2018). For a recent survey of the moving target search optimization literature, the reader is referred to Raap et al. (2019).

The Optimal Search Path with Visibility (OSPV) generalizes the classical OSP problem by taking into account the fact that a searcher can scan visible regions different from its location (Morin et al., 2010). The introduction of the possibility to search from a distance adds realism to the model, albeit at the price of an increased solution space size. Nonetheless, it is a crucial assumption for achieving models that reflect the way searching is actually conducted. For example, inter-region visibility is already taken into account to evaluate predefined search patterns in operational maritime SAR decision support systems such as SAR Optimizer (Abi-Zeid et al., 2019) and SAROPS (Kratzke et al., 2010). However, these systems do not currently propose nor optimize search paths. The research presented here is a step in that direction.

3. Optimal Search Path with Visibility: Problem Formulation

The OSPV problem is a single-sided detection search for an object moving in a discrete environment of N regions according to a Markovian model. The goal is to construct a search plan over T time steps that maximizes the probability of finding the object under the searcher's visibility and accessibility constraints. A search plan consists of a discrete path and Q discrete effort allocations at each time step. A unit allocation of search effort is an observation (a scan) by the searcher from its location to a visible region, possibly its own location. A searcher may actually consist of a team of agents following a common path while scanning one or many different regions simultaneously.

Let $\mathcal{T} = \{1, ..., T\}$ be the set of time steps, $\mathcal{R} = \{0, \dots, N-1\}$ the set of regions in the search environment, $\mathcal{Q} = \{0, \dots, Q\}$ the set of possible effort allocations (scans) that may be assigned to visible regions in one time step, and $A : \mathcal{R} \to 2^{\mathcal{R}}$ and $V : \mathcal{R} \to 2^{\mathcal{R}}$ the searcher's accessibility and visibility maps. The searcher's position at a time $t \in \mathcal{T}$ is $y_t \in \mathcal{R}$ and its effort allocation in region r at $t \in \mathcal{T}$ is $e_t(r) \in \mathcal{Q}$. A search plan P is defined by

$$P = \langle Y, E \rangle = \langle [y_1, y_2, \ldots, y_T], [e_1, e_2, \ldots, e_T] \rangle , \tag{1}$$

where

$$\begin{aligned} y_t &\in A(y_{t-1}) , & \forall t &\in \mathcal{T} , & (2)\\ e_t(r) > 0 &\Rightarrow r \in V(y_t) , & \forall t &\in \mathcal{T}, \forall r \in \mathcal{R} , & (3)\\ \sum_{r \in \mathcal{R}} e_t(r) &= Q , & \forall r &\in \mathcal{T} , & (4)\\ e_0(r) &= 0 , & \forall r &\in \mathcal{R} . & (5) \end{aligned}$$

Eq. (2) indicates that the searcher may only move to an accessible region and Eq. (3) indicates that effort can be allocated only to a region visible from the searcher's location. The total search effort allocated in one time step is equal to Q (Eq. (4)), the amount of available search effort per time step, and no search is conducted at time 0 (Eq. (5)). The search P is *feasible* iff it respects Eqs. (1)–(5). A feasible search plan P is *optimal* iff it maximizes a performance measure, the *cumulative overall probability of success* or *COS* (Eq. (12)). In order to understand this performance measure, we define three types of events: presence, detection and motion.

A *presence* event C_t^r occurs when the object is located (but not necessarily detected) in region r at time t. A *motion* event M_t^{sr} occurs when the object moves from region s to region r at time t. A *detection* event D_{tq}^{sr} occurs when an allocation of q scans from region s detects an object located in r at time t[1]. Under the assumption that searchers are located in the same region, we refer to the detection event as D_t^r without loss of generality. Since the OSPV is a detection search problem, the search will stop as soon as a detection occurs.

Motion Model. The object's motion model is assumed to be stationary Markovian. It is described by a matrix $\mathbf{M}$ where $\mathbf{M}(s, r)$ is the probability of an object moving from region s to region r

[1]No false detections are taken into account in the OSPV problem formalism.

within one time step. For all $t \in \mathcal{T}$, we have

$$\mathbf{M}(s,r) \overset{\text{def}}{=} \Pr\left\{\mathsf{M}_t^{sr}\right\} . \tag{6}$$

Detection Model. Given $s, r \in \mathcal{R}$, $t \in \mathcal{T}$ and $q \in \mathcal{Q}$, $pod_t(s,r,q)$ is the conditional probability of a detection event in region r at time t when a searcher in region s assigns q scans to region r. This probability, conditional to the object's presence in r, is defined as follows:

$$pod_t(s,r,q) \overset{\text{def}}{=} \Pr\left\{\mathsf{D}_t^r \mid \mathsf{C}_t^r\right\} . \tag{7}$$

In practice, a *pod* is derived from a sensor's (e.g., a visual search) characteristics under given environmental conditions, as a function of a given search object type at a given range (Frost, 1999). In the OSPV, we make the common assumption that the detection law is exponential (Stone, 2004) defined by

$$pod_t(s,r,q) = 1 - \exp\left(-W_t(s,r) \times q\right), \tag{8}$$

where $W_t(s,r)$ is the sweep width, a detectability index of the object (defined in Section 5.4 since it is instance-specific). The exponential detection law provides a lower bound for detection probabilities obtained with other detection laws (Frost & Stone, 2001).

Presence model. The *a priori* knowledge on the object's presence in region r is defined as

$$poc_0(r) \overset{\text{def}}{=} \Pr\left\{\mathsf{C}_0^r\right\} , \tag{9}$$

where $\Pr\left\{\mathsf{C}_0^r\right\}$ is the prior probability that the object is in region r before the search is initiated. For $r \in \mathcal{R}$ and $t \in \mathcal{T}$, $poc_t(r)$, the joint probability of the object arriving at t from any region s and not being detected in region s before t (also called the probability of containment), is

$$poc_t(r) \overset{\text{def}}{=} \Pr\left\{\mathsf{C}_t^r\right\} = \sum_{s \in \mathcal{R}} \Pr\left\{\mathsf{M}_{t-1}^{sr}\right\} \Pr\left\{\mathsf{C}_{t-1}^s\right\} \left(1 - \Pr\left\{\mathsf{D}_{t-1}^s \mid \mathsf{C}_{t-1}^s\right\}\right) . \tag{10}$$

Performance measures. The probability of finding the object in region r at time t is $pos_t(r)$, the local probability of success. It is the joint probability of a presence event and of a detection event:

$$pos_t(r) \stackrel{\text{def}}{=} \Pr\{\mathsf{C}_t^r \cap \mathsf{D}_t^r\} = \Pr\{\mathsf{C}_t^r\} \times \Pr\{\mathsf{D}_t^r \mid \mathsf{C}_t^r\} \ . \tag{11}$$

The objective is to maximize the total probability of success across all regions and time steps, i.e., the *cumulative probability of success* of a search plan. Since only a single success is possible in a detection search, we define this probability as

$$COS(P) = \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} pos_t(r) \, , \tag{12}$$

where the local probability of success is

$$pos_t(r) = poc_t(r) \, pod_t(y_t, r, e_t(r)) \, . \tag{13}$$

Following an unsuccessful search at time $t-1$, the updated probability of containment at time t is given by:

$$poc_t(r) = \sum_{s \in \mathcal{R}} \mathbf{M}(s, r) \left[poc_{t-1}(s) - pos_{t-1}(s)\right] \ . \tag{14}$$

A MILP for the OSPV problem. A MILP can be formulated for the OSPV where the objective is to find a search plan maximizing Eq. (12). This is possible since the *pod* function, the poc_0 distribution, the Markovian motion matrix $\mathbf{M}$ and the initial searcher's position y_0 are all known. As a consequence, the *poc* update equation, i.e., Eq. (14), can be linearized. Binary decision variables and constraints can be used to define the search plan (Eqs. (1), (2) and (3)) to optimize under effort constraints (Eqs. (4) and (5)). Continuous variables can be used to keep trace of the probability models (Eqs. (8), (12) and (14)). The complete MILP model for the single scan case ($Q = 1$) and its extension to the case of multiple scans ($Q \geq 1$) can be found in Morin et al. (2009) and Morin et al. (2010) respectively.

4. Using Ants for Search Path Planning with Visibility

Ant colony optimization, or ACO, is a metaheuristic optimization technique that probabilistically constructs and updates a population of candidate solutions to a problem based on a common "memory", called the pheromone trails. At each cycle (iteration) of an ACO algorithm, each ant in the colony builds a single solution by selecting components (unit parts of a solution) according to the shared knowledge of components' quality, stored in the pheromone trails. For instance, in our case, there are two types of solution components, a destination r at time t and an effort allocation to one or many regions at time t. The probability of an ant choosing a given component to be part of its solution (called the *transition probability*) is a function of the component's pheromone value and, when available, of the value associated to the component by a heuristic, called *heuristic information.* An ACO algorithm *cycle* consists of stochastically generating one solution per ant using the trails (the *generation process*), and of updating these trails (the *update process*) as a function of the pheromone model, which provides an *update rule* dictating which and when the generated candidate solutions are used to update the trails, and an *update equation* to quantify a solution component's quality.

Various strategies can help ACO algorithms converge towards high quality solutions namely intensification, diversification, and restarts. *Intensification* is the process by which ants converge to better solutions whereas *diversification* is the process by which stagnation, i.e., numerous cycles without improvement when such improvement is possible, is avoided (Hoos & Stützle, 2004). *Restarts* are a widely used diversification mechanism that proved to be successful in a variety of metaheuristics (Hoos & Stützle, 2004). They enable the algorithm to reinitialize its solving process in the absence of local improvements. One particular intensification mechanism is based on *pheromone boosting* introduced in Solnon (2002). It could be seen as a diversification mechanism or as an intensification mechanism depending on the way it is used. An exhaustive treatment of ACO algorithms can be found in López-Ibáñez et al. (2016).

In order to solve the OSPV, we developed the ASPV algorithm, based on ACO principles, and defined 96 algorithm variants based on four main components of traditional ACO: pheromone initialization, pheromone update, with or without restarts, and with or without boosting (Table 1). For clarity purposes, whenever a variant is named using solely the acronyms defined in Table 1, they are listed in the following order: pheromone initialization, pheromone update, restart, and boosting.

Table 1: Variant components with acronyms, names and section numbers

Type	**Name**	**Section**	**Name**	**Section**
Initialization	iU: Uniform	4.2	iO: OSPV	4.2
	iR: Random	4.2		
Updates	uAA: All-Ants	4.5	uGB: Global-best	4.5
	uIB: Iteration-best	4.5	uORBU: On restart-best upd.	4.6
	uRB: Restart-best	4.6	uOGBU: On global-best upd.	4.5
Restarts	rG: Geometric	4.6	rN: Without	4.6
	rL: Luby	4.6		
Boosting	bY: With	4.3	bN: Without	4.3

4.1. ASPV Algorithm Main Routine

Algorithm 1 outlines the main routine of ASPV. Functions `Initialize()`, `Generate()`, and `Update()` are variant-dependent placeholders. Given an OSPV problem instance *Ospv*, the total number C of candidate solutions to generate at each cycle and the pheromone evaporation rate ρ, used in order to avoid stagnation in a local optimum, the algorithm first initializes the pheromone trails using the function `Initialize()` in two pheromone tables: one $T \times N$ table τ^{path} for the pheromone on the move components, and one $T \times N$ table τ^{eff} for the pheromone on effort units allocation components. Whenever needed, pheromone values normalization is carried out for each table independently so that the sum of their pheromone values equals 1. In some variants, boosting is also performed during initialization.

In each cycle, the function `Generate()` is used to construct a new candidate solution set $\mathcal{C}$ based on the pheromone trails. Then, the function `Update()` is used to update τ^{path}, τ^{eff}, and the incumbent (best-so-far) solution P^{best}. Depending on the variant, in some cycles, a restart may occur. The stopping criterion is based on a time limit.

4.2. Initialize Pheromones

In most ACO algorithms, pheromone trails are initialized to the same values (Dorigo & Stützle, 2019). We call this first variant, where all initial pheromone values are equal, *uniform pheromone initialization* (iU). In order to ensure a strong diversification between restarts, we introduce a *random pheromone initialization* (iR) variant where the initial pheromone values are generated from a random uniform distribution between 0 and 1. In both variants, the pheromone values

Algorithm 1: ASPV($Ospv$, C, ρ)

```
Input: An OSPV problem $Ospv$, the size of the colony $C$, and the evaporation rate $\rho$.
Output: The incumbent search plan $P^{\text{best}}$.
begin
    $\tau^{\text{path}}$, $\tau^{\text{eff}}$ ← Initialize() ;
    while stopping criterion is not met do
        $\mathcal{C}$ ← Generate() ;
        $P^{\text{best}}$, $\tau^{\text{path}}$, $\tau^{\text{eff}}$ ← Update() ;
    end
    return $P^{\text{best}}$ ;
end
```

are normalized so that their sum in each table is equal to 1. Both uniform (iU) and random (iR) pheromone initialization are generic. However, we hypothesize that problem-specific knowledge can help the algorithm in finding a high quality solution. We therefore introduce an *OSPV-based pheromone initialization* procedure (iO). In this variant, the algorithm fixes the pheromone values of each region s and of each region r visible from s as follows:

$$\tau_{ts}^{\text{path}} = \sum_{r \in \mathcal{R}} pocm_t(r) pod_t(s, r, 1) , \tag{15}$$

$$\tau_{ts}^{\text{eff}} = pocm_t(s) , \tag{16}$$

where $pocm_t(r)$ is the probability that the search object reaches region r at step t in the absence of search with $pocm_0 = poc_0$:

$$pocm_t(r) = \sum_{s \in \mathcal{R}} \mathbf{M}(s, r) pocm_{t-1}(s) . \tag{17}$$

It should be noted that, in Eq. (15), $pod_t(s, r, 1)$ is the probability of detection for a single scan from region s to region r at time t. As a result, the pheromone trails in τ_{ts}^{path} correspond to the overall probability of success at a time t in a region s assuming a single unit of effort can be allocated to each visible region and that no search has been done prior to time step t. For the τ_{ts}^{eff} value, we simply assume that no searches take place. Although this is not true in the OSPV problem context, the information on the object's motion initially embedded in the trails can still be important during the solving process. Once all values have been set using Eqs. (15)

and (16), a small randomly generated epsilon value is added to the trail values before normalization. This ensures that a region r with low $pocm_t(r)$ still has a probability of being chosen by an ant. Furthermore, it favors diversification as it gives a little less or a little more importance to Eqs. (15) and (16).

4.3. Boost Pheromone Initialization Using a Greedy Heuristic

Pheromone boosting basically consists of initializing the pheromone values to promising values. As such, it is not unlike the OSPV-based pheromone update (iO). The idea of such a preprocessing step was introduced and illustrated in Ant-Solver which was designed to solve constraint satisfaction problems (Solnon, 2002). Some form of pheromone boosting, or preprocessing, was also applied in other problems such as the traveling salesperson problem (TSP), e.g., Dai et al. (2009) used a minimum spanning tree of the TSP graph to initialize the pheromone.

Our approach to boosting is to use a problem-specific greedy heuristic to find a search plan P^{boost} to perform a first update of the trails right after initialization. This can be seen as an intensification mechanism that directs the ants towards a promising subspace of the solution space. To greedily construct $P^{\text{boost}} = \langle Y, E \rangle$, the algorithm first chooses the accessible region with the highest overall success probability as the new searcher's destination. That is, it evaluates each possible destination by allocating each effort unit greedily to visible regions from that destination, i.e., it maximizes the local success by scanning one visible region at a time. Then it selects the accessible region where the overall success, i.e., the sum of all local successes, is the highest. Such a sequential effort allocation is possible, since we use an exponential detection law making the detection process memoryless. When P^{boost} is found, it is used to update the pheromone trails using a given evaporation rate ρ^{boost}. Although ρ^{boost} and ρ could be different, we use the same values to avoid an additional parameter in the algorithm.

We use bY to denote variants with boosting and bN to denote variants without.

4.4. Generate Candidate Solutions

Candidate solutions are generated at each cycle of the algorithm by the `Generate()` function, which constructs C candidate solutions. A candidate solution $P^{\text{cand}} \in \mathcal{C}$ consists of a searcher's path and a sequence of effort allocations. At each time step $t \in \mathcal{T}$, the ant chooses a feasible searcher's move from y_{t-1} to y_t and defines the allocation vector e_t by distributing Q effort units to visible regions from y_t. Each part of a solution, or solution component, is chosen by an ant with some transition probability depending on the pheromone values associated with the component.

Let $P^{\text{cand}}.y_t \leftarrow r$ and $P^{\text{cand}}.e_t(r)_{++}$ be the solution components for moving to region r and for allocating an additional effort unit to region r in search plan P^{cand} at time t. A feasible searcher's move $P^{\text{cand}}.y_t \leftarrow r$ is chosen with a transition probability of

$$p_{P^{\text{cand}}.y_t \leftarrow r} = \frac{\tau_{tr}^{\text{path}}}{\sum_{r' \in A(y_{t-1})} \tau_{tr'}^{\text{path}}} , \tag{18}$$

and Q feasible effort allocations $P^{\text{cand}}.e_t(r)_{++}$ are chosen with a transition probability of

$$p_{P^{\text{cand}}.e_t(r)_{++}} = \frac{\tau_{tr}^{\text{eff}}}{\sum_{r' \in V(y_t)} \tau_{tr'}^{\text{eff}}} . \tag{19}$$

Solutions are built by adding one component at a time, first a feasible searcher's move, then Q feasible effort allocation and so on. Once C solutions have been generated, the `Generate()` function returns and the algorithm launches the update process.

4.5. Update Pheromones

The `Update()` function updates the pheromone trails and is variant-independent in our ASPV algorithm. However, the "when" is variant-dependent. That is, the solutions used for the update and the cycle during which the update takes place depend on the algorithm variant. Whenever a solution P is used to update the pheromone trails, τ_{tr}^{path} and τ_{tr}^{eff} are modified as follows:

$$\tau_{tr}^{\text{path}} \leftarrow \tau_{tr}^{\text{path}} + \frac{\rho}{S}\left(os_t + \frac{COS(P)}{T}\right) , \tag{20}$$

$$\tau_{tr}^{\text{eff}} \leftarrow \tau_{tr}^{\text{eff}} + \frac{\rho}{S}\left(\sum_{r \in \mathcal{R}} pos_t(r) + \sum_{r \in \mathcal{R}} \frac{P.e_t(r)COS(P)}{QT}\right) , \tag{21}$$

where S is a normalization factor that depends on how many updates were carried out in the cycle, and os_t is the overall success at a time step t, i.e., $os_t = \sum_{r \in \mathcal{R}} pos_t(r)$. After all updates have been carried out, we use the evaporation rate ρ to decrease the pheromone values as follows:

$$\tau_{tr}^{\text{path}} \leftarrow (1-\rho)\tau_{tr}^{\text{path}} , \tag{22}$$

$$\tau_{tr}^{\text{eff}} \leftarrow (1-\rho)\tau_{tr}^{\text{eff}} . \tag{23}$$

We defined a total of six variants of pheromone update schemes. Four can be used with and without restarts: The first scheme is an all-ants pheromone updates procedure (uAA) that is reminiscent of ant systems (Dorigo et al., 1996). In this setting, all candidate solutions generated in the cycle are used to update the pheromone trails, and updates are performed every cycle. The second scheme is an iteration-best pheromone update (uIB) procedure where only the best candidate solution of each cycle is used for the update. The third scheme, global-best updates (uGB), uses the best-so-far incumbent solution to update the pheromone at each cycle. All three variants are discussed in recent literature on ACO (López-Ibáñez et al., 2016). Finally, we have an "on global-best updates" (uOGBU) scheme consisting in updating the pheromone only when the best candidate solution of an iteration is better than the best-so-far (global-best) incumbent solution. Two further pheromone update schemes are used only with restarts and are further discussed in Section 4.6.

4.6. Update Pheromones with Restarts

Restarting is a frequent strategy used in solvers, especially in Boolean satisfiability problems solvers (Audemard & Simon, 2012) and in constraint optimization or constraint satisfaction problems solvers (Wu & van Beek, 2007). A restart is used when a specific number of cycles without improvement of the best incumbent since the last restart is reached. Generally, the number of cycles without improvement follows a sequence of constants $\bar{R} = \langle \bar{r}_0, \bar{r}_1, \ldots \rangle$ called the restart strategy, where $\bar{r}_i \in \mathbb{N}$ is the total number of allowed cycles without improvement following i restarts.

Universal, problem independent, restart strategies are often used in solvers, e.g., Gecode (Schulte et al., 2019). We used two such strategies: a geometric sequence and a Luby sequence. The *geometric restart strategy* was notably tested by Walsh (1999) and provided good results. It consists of a sequence such that $\bar{r}_i = cb^i$ where c is a chosen constant and b is a chosen base. The *Luby restart strategy* (and Luby sequence) was introduced by Luby et al. (1993) as an optimal universal restart strategy for Las Vegas algorithms. Starting at 1, the entire sequence is repeated and a new value corresponding to double that of the last value is introduced at the end of the sequence:

$$\underbrace{1,}_{\text{init}}\ \underbrace{1,2,}_{\text{step 1}}\underbrace{1,1,2,4,}_{\text{step 2}}\underbrace{1,1,2,1,1,2,4,8,}_{\text{step 3}}1,\ \ldots \tag{24}$$

In practice, the terms of the Luby sequence can be multiplied by a factor to allow for more time between restarts. When no restarts are used by a given variant, the acronym rN is used in the variant name. Geometric restarts are denoted by the rG acronym. Luby restarts are denoted by the rL acronym.

In the ASPV algorithm context, a restart is a reinitialization of the pheromone trails, which varies as a function of the pheromone initialization scheme and of whether boosting is used or not. A distinction is made, when restarts are used, between the best-so-far and the restart-best solution. We call "global-best solution" the best incumbent across all restarts and "restart-best solution" the best incumbent since the last restart. This distinction is important for the algorithm variants; when using the global-best updates (uGB) or the "on global-best updates" (uOGBU), the best incumbent across all restarts is used to update the trails (either at each cycle for a uGB variant or when a better solution is found for a uOGBU variant).

Two other pheromone update variants use the best incumbent since the last restart. That is, the restart-best (uRB) and the "on restart-best updates" (uORBU) pheromone update procedures, which respectively corresponds to global-best (uGB) and "on global-best updates" (uOGBU) using the restart-best solution instead of the global-best solution.

5. Experiments

In this section, we describe the experiments conducted to evaluate the 96 proposed ACO algorithm variants in order to determine the best variant. These experiments consisted of three phases: The *configuration phase* enables us to find the best parameter pairs for each variant. The *"multi-runs" evaluation phase* (M-Eval) enables us to determine how the performance of an ACO varies between runs on a given instance. The *"across-instances" evaluation* (A-Eval) phase provides a better understanding of an ACO performance on a variety of instances as reported in Birattari (2004). Finally, we compare the results of the best ACO variant with the MILP model described in Section 3 and with the greedy heuristic described in Section 4.3, hereinafter named Greedy.

5.1. ASPV Configuration and Evaluation

In order to configure each of the 96 variants, we first generated a total of 100 unique pairs of evaporation factors ρ and colony sizes C from a uniform distribution in the interval $[0.001, 0.1]$ (up to four decimal places) and in $[10, 1000]$ respectively. The 100 generated parameters pairs, which that cover a wide range of values, are displayed in the supplementary material. For rG variants using a geometric restart strategy, we chose $c = 1$ and $b = 2$ as parameters. The obtained geometric sequence grows fast even with such small values, which results, very quickly, in long runs without restarting. For rL variants involving Luby restarts, each term in the sequence was multiplied by 256 in order to avoid a too short time interval between restarts. Nonetheless, the resulting Luby sequence grows more slowly than the geometric sequence.

During the *configuration phase*, each of the 96 algorithm variants was run using each of the 100 parameter pairs on 50 different instances of the OSPV problem with varying complexity as described in Section 5.4 (a total of 5,000 runs per variant). For each variant, the parameter pair with the highest average performance across all instances was deemed the best for that variant and was kept for the evaluation phase. In the M-Eval phase, each algorithm variant, configured according to the best parameter pair, was run on another 50 instances not seen during the configuration phase 30 times (a total of 1,500 runs per variant). Furthermore, for comparison purposes, the greedy heuristic from Section 4.3 and the MILP model of the OSPV problem from Section 3 were also run on each of the 50 M-eval problem instances. In the A-Eval phase, the five best ACO algorithms identified in the M-Eval phase, were run once on 1,500 new instances (7,500 runs in total). The A-Eval benchmark consisted of 30 instances for each pair of horizon T and number of scans Q described in Section 5.4. The average performance of the ACO was compared to that of Greedy on this particular benchmark. The MILP model was excluded from the A-Eval phase since it did not fare well on the largest instances of the M-Eval phase.

The framework used to generate the problem instances and to run the algorithms was developed using the C++ programming language. The experiments of the configuration phase and of the M-Eval phase were run in parallel using GNU Parallel (Tange, 2011) on Intel Xeon Gold 6148 Skylake (2.4GHz) CPUs. Up to 8 GB of memory were allowed per core for the MILP solver. Due to the unavailability of the Skylake CPUs, the experiments of the A-Eval phase were run in parallel using GNU Parallel (Tange, 2011) on an AMD Ryzen 9 5900X CPU. The problem instances and the allowed solving time for each instance size are specified in Section 5.4.

5.2. Performance metrics

In order to evaluate the performances of the algorithm variants, we use the relative cumulative overall probability of success of a search plan P, normalized as a function of the minimum and the maximum cumulative probabilities of success attained, by the considered variants or algorithms, in the allowed time.

For the configuration phase, the performance metric is defined as follows:

$$RCOS^{\text{conf}}(P^i_{c,v}, \mathcal{P}^i_v) = \begin{cases} \frac{COS(P^i_{c,v}) - \min\limits_{P' \in \mathcal{P}^i_v} COS(P')}{\max\limits_{P' \in \mathcal{P}^i_v} COS(P') - \min\limits_{P' \in \mathcal{P}^i_v} COS(P')}\,, & \text{if } \max\limits_{P' \in \mathcal{P}^i_v} COS(P') \neq \min\limits_{P' \in \mathcal{P}^i_v} COS(P')\,; \\ 1\,, & \text{otherwise}\,, \end{cases} \tag{25}$$

where $P^i_{c,v}$ is the search plan obtained by configuration c of variant v on instance i. Recall that we have 100 configuration pairs, 96 variants and 50 instances. $\mathcal{P}^i_v$ is the set of 100 plans corresponding to the 100 parameter configurations of variant v run on instance i. We then compute, for each configuration c of variant v, $\overline{RCOS^{\text{conf}}}(c, v)$, the average of $RCOS^{\text{conf}}(P^i_{c,v}, \mathcal{P}^i_v)$ over the instances i. The configuration c with the highest $\overline{RCOS^{\text{conf}}}(c, v)$ is selected to be used for variant v in the evaluation phases (M-Eval and A-Eval).

For the M-Eval phase where we select the best ACO variant, the performance metric is defined as follows:

$$RCOS^{\text{eval}}(P^{i,j}_v, \mathcal{P}^i) = \begin{cases} \frac{COS(P^{i,r}_v) - \min\limits_{P' \in \mathcal{P}^i} COS(P')}{\max\limits_{P' \in \mathcal{P}^i} COS(P') - \min\limits_{P' \in \mathcal{P}^i} COS(P')}\,, & \text{if } \max\limits_{P' \in \mathcal{P}^i} COS(P') \neq \min\limits_{P' \in \mathcal{P}^i} COS(P')\,; \\ 1\,, & \text{otherwise}\,, \end{cases} \tag{26}$$

where $P^{i,j}_v$ is the search plan obtained by the j^{th} run of variant v on instance i. Recall that we have 30 runs per instance and variant, and 96 variants. $\mathcal{P}^i$ is the set of 2880 plans obtained for any variant on instance i. We then compute, for each variant v, $\overline{\overline{RCOS^{\text{eval}}}}(v)$, the average of $RCOS^{\text{eval}}(P^{i,j}_v, \mathcal{P}^i)$ over the runs j and the instances i. The ACO variant with the highest value of the metric $\overline{\overline{RCOS^{\text{eval}}}}(v)$ is selected as the best ACO variant.

For the comparison between the best ACO variant, the MILP solved by IBM ILOG CPLEX 12.9 (hereinafter referred to as CPLEX) and Greedy during the M-Eval phase, the performance metric of Eq. 26 is used. However, in that equation, the set of search plans $\mathcal{P}^i$ includes the COS of the benchmark methods, i.e., Greedy and the best performing configuration of CPLEX (we describe the tested configurations in Section 5.3). Still, the metric used for comparison purposes is $\overline{RCOS^{\text{eval}}}(i, v)$, the average of $RCOS^{\text{eval}}(P^{i,j}_v, \mathcal{P}^i)$ over the runs j.

For the A-Eval phase where we further compare the best ACOs from the M-Eval phase with

Greedy, we used the $RCOS^{\text{eval}}(P_v^{i,j}, \mathcal{P}^i)$ metric from Eq. 26, and we computed the mean signed relative COS difference between variant v and Greedy as follows:

$$MSDG(v) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \left(RCOS^{\text{eval}}(P_v^{i,j}, \mathcal{P}^i) - RCOS^{\text{eval}}(P_{\text{Greedy}}^{i,j}, \mathcal{P}^i) \right) , \tag{27}$$

where $\mathcal{P}^i$ contains the COS of our five best ACO variants and of our greedy heuristic for instance i, $j = 1$ since there is a single run of each algorithm (ACO variants and Greedy) per instance, and set $\mathcal{I}$ is a set of instances, e.g., 30 instances with a given time horizon T and number of scans Q (as described in Section 5.4).

5.3. MILP Solver Configuration

One of our objectives was to compare the best ACO variant with an MILP formulation (described at the end of Section 3). We solved the MILP model using ILOG CPLEX 12.9. It is well known that the performance of CPLEX depends on its configuration and that the default configuration performs well on a variety of problems. In fact, when conducting a search for a solution, a MILP solver, such as CPLEX, builds a search tree. Each node of the search tree is either a partial solution or a subproblem (depending on the exact algorithm used by the solver). Complete solutions are found at the leaves of the tree. The node selection strategy that guides the solver in the selection of the next node to explore, i.e., the next subtree, can therefore influence the search process. Another element that is important in configuring the CPLEX algorithm is the MIP emphasis parameter which biases the solver towards finding feasible solutions or proving the optimality of the current incumbent solution.

Our experiment was designed with six CPLEX configurations: the *default* configuration, *emphasis on feasibility*, *emphasis on optimality*, default configuration with a *scaling* of the probabilities by a factor of 10^5, *best estimate node selection*, and *depth-first node selection*. Since the performance of CPLEX is meant as a benchmark, we include in $\mathcal{P}^i$, for the computation of $\overline{RCOS^{\text{eval}}}(i, v)$ during the M-Eval phase, the highest COS value among the 6 CPLEX configurations for each of the 50 instances i.

5.4. Generated Search Environments Characteristics

In order to generate OSPV problem instances, the search environment was represented by a grid of l cells by l cells. The accessibility map was defined by the distance that a searcher can travel in one time step, i.e., an accessibility radius $a^{\max}$, and the visibility map was defined by the maximal effort allocation range, i.e., a visibility radius $v^{\max}$:

$$(r \in A(s) \Leftrightarrow \text{dist}(s,r) \leq a^{\max}) \ , \qquad \forall s, r \in \mathcal{R} \, , \tag{28}$$

$$(r \in V(s) \Leftrightarrow \text{dist}(s,r) \leq v^{\max}) \ , \qquad \forall s, r \in \mathcal{R} \, , \tag{29}$$

where $\text{dist}(s,r)$ is the distance between s and r in distance units. For our instances, the detectability index (W) from Eq. 8 is such that

$$W_t(s,r) = \begin{cases} \frac{v^{\max} - \text{dist}(s,r)}{\text{area}(r)} & \text{if } r \in V(s) \, ; \\ 0 & \text{if } r \notin V(s) \, , \end{cases} \qquad \forall t \in \mathcal{T}, \forall s, r \in \mathcal{R} \, , \tag{30}$$

where $\text{area}(r)$ is the area of region r (in square distance units). We assume that the detectability of the object in a region r decreases as the distance between the current searcher's region s and r increases. This detectability becomes 0 when the maximum visibility range $v^{\max}$ is reached. The initial searcher's position y_0 is randomly generated from a uniform distribution over regions. The $poc_0(t)$ distribution is obtained from a uniform random distribution over the interval $[0,1]$ and then normalized to obtain a sum of 1. The probability that the object moves from a region s to an accessible region r is obtained from a uniform random distribution on the interval $[0,1]$ whereas it is null to any non-accessible region r'.

We generated instances of increasing complexity, namely grids of l by l cells with $l \in \{2, 3, \ldots, 11\}$ for a number of regions $N \in \{2^2, 3^2, \ldots, 11^2\}$, where the time horizon $T = N$. In comparison, instances found in the literature for similar problems related to search path planning use a horizon of 20 steps (Perez-Carabaza et al., 2018) or of up to 40 steps (Sato & Royset, 2010). In addition, these do not take into account the visibility dimension like we do, which increases the size of the search space of an algorithm by a factor representing the number of feasible allocations of Q units of effort growing exponentially in T.

For the configuration and the M-Eval phases, we generated, for each grid size, an instance with $Q \in \{1, 2, \ldots, 5\}$ and we set $T = N$ (50 different instances per phase). For the A-Eval phase, we generated 30 instances per pair of horizon T and number of scans Q (1,500 instances). The allowed solving time was set per instance as a function of the time horizon T (Table 2).

Table 2: Solving wall clock time in seconds for each time horizon T

T	**Allowed time (sec.)**	T	**Allowed time (sec.)**
4	60	49	720
9	120	64	960
16	240	81	1200
25	360	100	1500
36	540	121	1800

6. Results and Discussion

We present and discuss the M-Eval phase results where each algorithm variant was applied to 50 previously unseen instances (30 runs per instance) using their best configuration parameters as determined in the configuration phase described in the supplementary material. We then compare the five best ASPV variants of the M-Eval phase to Greedy on 1,500 new problem instances for the A-Eval phase and report the results of the best variant. The results of the remaining four variants are included in the supplementary material.

6.1. Comparison Between ACO Variants (M-Eval Phase)

Figure 1 shows, for each ASPV variant, the distribution of the relative COS, $RCOS^{\text{eval}}(P_v^{i,j}, \mathcal{P}^i)$, over 30 runs of each of the 50 instances. Variants are ordered in decreasing order of average $RCOS^{\text{eval}}(P_v^{i,j}, \mathcal{P}^i)$, $\overline{\overline{RCOS^{\text{eval}}}}(v)$, across all instances which is displayed as a blue diamond. For each variant, the box displays the first quartile, the median, and the third quartile. Whiskers extends from the first (resp. the third) quartile to the lowest value not smaller (resp. highest value not larger) than 1.5 times the interquartile range from the first quartile down (resp. third quartile up). Outliers are represented as black dots.

We notice that, except for the variant in position 20, only variants using the OSPV pheromone initialization procedure (iO) made it to the top 20 performers (considering the $\overline{\overline{RCOS^{\text{eval}}}}(v)$ performance criterion). This supports the implementation of a problem-specific pheromone initialization, iO, for the OSPV.

The fact that the boosting procedure (bY) is used in 4 of the top 5 variants also supports this point since boosting, in our terminology, involves a first round of pheromone updates using a solution from a greedy procedure. Boosting, nonetheless, appears to have a lesser impact than problem specific pheromone initialization (iO).

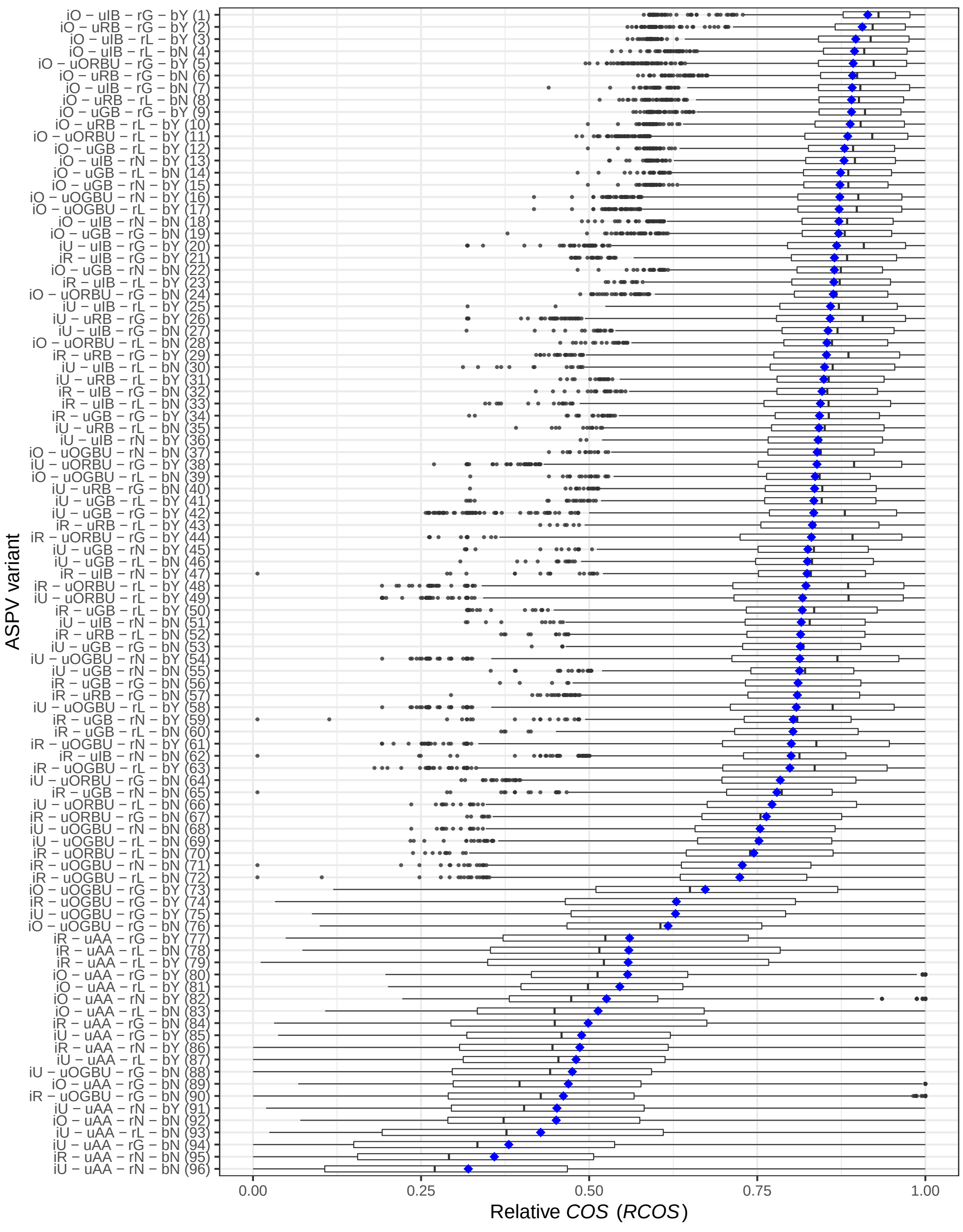

Figure 1: Distribution of the $RCOS^{\text{eval}}$ values (calculated on a per instance basis) for all 30 runs of each ASPV variant on all instances; blue diamonds represent the per variant average $\overline{\overline{RCOS^{\text{eval}}}}(v)$ (M-Eval phase)

Both Luby (rL) and geometric (rB) restarts, are used in 16 out of the top 20 variants, which illustrates their benefit as a diversification mechanism in our context.

As for the worst performers, the "all ants" (uAA) pheromone update procedure is over-represented in the last quartile (the 24 worst performers). In fact, 18 out of 24 of the worst performers use "all ants" (uAA) pheromone updates. The other worst performers use the "on updates of the global-best" (uOGBU) pheromone update. We notice that uAA is the least restrictive update procedure (all other things being equal, updates are performed the most frequently), whereas uOGBU pheromone update is the most restrictive (all other things being equal, updates are performed less frequently). Some uOGBU variants, however, did perform well and made it to the top 20. Those "good" uOGBU variants are two variants with OSPV-based pheromone initialization (iO) and boosting (bY). The 16th best variant employed Luby restarts (rL). However, the restart procedure in the context of an uOGBU does not give many opportunities for pheromone updates; the global-best solution, used to decide when updates are performed, does not change between restarts. The 16th and 17th best variants both used iO and bY, two problems specific pheromone initialization (including boosting). As a result, it appears best, in the ASPV, to avoid the two extreme update procedures (uAA and uOGBU) unless a mechanism for better convergence is implemented. In fact, the various middle-of-the-road approaches to trails updating, namely iteration-best (uIB), restart-best (uRB), global-best, and "on updates of the restart-best" (uORBU), all performed well in general.

Finally, regarding the spread and distribution of the global performance of variants based the $\overline{\overline{RCOS}}^{\text{eval}}(v)$ metric, we notice that the worst variant, iU – uAA – rN – bN, has an average performance of .32 whereas the best performer, iO – uIB – rG – bY, has an average performance of .914. A total of 13 variants have an average performance across all instances lower or equal to .5, 21 variants have an average performance in the interval $(.5, .8)$, 32 in $(.8, .85]$, 28 in $(.85, .9]$, and only 2 have an average performance above .9.

6.2. Benchmarking ACO Variants Against a MILP and a Greedy Approach (M-Eval Phase)

In light of the results presented in Section 6.1, we retained the iO – uIB – rG – bY variant for comparison with the MILP and Greedy results in the M-Eval phase.

Table 3 shows the performance of the three considered approaches, grouped by instance, for a subset of instances (complete results are included in the supplementary material). Each cell

Table 3: Average relative COS, $\overline{RCOS^{\text{eval}}}(i, \text{iO} - \text{uIB} - \text{rG} - \text{bY})$ of the top ASPV variant, $RCOS^{\text{eval}}(P^{i,b}_{\text{CPLEX}}, \mathcal{P}^i)$ of CPLEX for its best configuration b, and $RCOS^{\text{eval}}(P^{i,1}_{\text{Greedy}}, \mathcal{P}^i)$ for the single run of Greedy; for the ASPV algorithm, the 95% confidence interval over 30 runs around the mean is reported; italic font indicates that the method is dominant, but not strictly dominant; bold font indicates that the method is strictly dominant; the complete table is included in the supplementary material (M-Eval phase)

		Number of scans (Q)				
Method	T	1	2	3	4	5
iO - uIB - rG - bY	9	[*1*, *1*]	[.99, .99]	[*1*, *1*]	[**.97**, **.97**]	[**1**, **1**]
	25	[.96, .98]	[**.95**, **.97**]	[**.94**, **.96**]	[**.9**, **.92**]	[*.94*, *.96*]
	49	[**.97**, **.97**]	[**.96**, **.97**]	[**.96**, **.97**]	[**.97**, **.98**]	[**.95**, **.96**]
	81	[**.88**, **.90**]	[**.92**, **.94**]	[**.95**, **.96**]	[**.97**, **.98**]	[**.97**, **.98**]
	121	[**.82**, **.86**]	[*.87*, *.90*]	[**.95**, **.96**]	[**.92**, **.93**]	[**.94**, **.95**]
Greedy	9	.98	.95	.96	.89	.94
	25	.47	.89	.92	.89	*.96*
	49	.66	.68	.92	.93	.80
	81	.64	.35	.58	.81	.93
	121	.49	*.87*	.35	.80	.82
Best of CPLEX	9	*1*	**1**	*1*	.87	.88
	25	**1**	.35	.41	.30	–
	≥ 36	–	–	–	–	–

represents the result of 30 runs for ASPV per instance, the best run for CPLEX, or a single Greedy run. The results reported for the iO – uIB – rG – bY ASPV variant are the 95 % confidence intervals around the mean $\overline{RCOS^{\text{eval}}}(i, v)$ over 30 runs. For CPLEX, we report the best performing configuration for each instance. We performed a single run of Greedy since it is a deterministic algorithm without configuration parameters. Italic font indicates that the method, or algorithm, is dominant, but not strictly dominant: there exists another approach with an equivalent performance or we cannot conclude the difference is significant. Bold font indicates that the method is strictly dominant, we can conclude that it outperforms the other approaches.

We can see that, in most cases and especially on larger instances, the top ASPV variant outperforms both Greedy and the best CPLEX configuration in the allowed time. When T and Q are small, CPLEX has a good performance. It outperforms the metaheuristic in four cases. Of course, the benefits of using a metaheuristic such ACO is often on the largest instances, since those are the realistic ones. In our case, CPLEX cannot find a good initial solution in the allowed time for instances where $T \geq 36$ (represented by a dash).

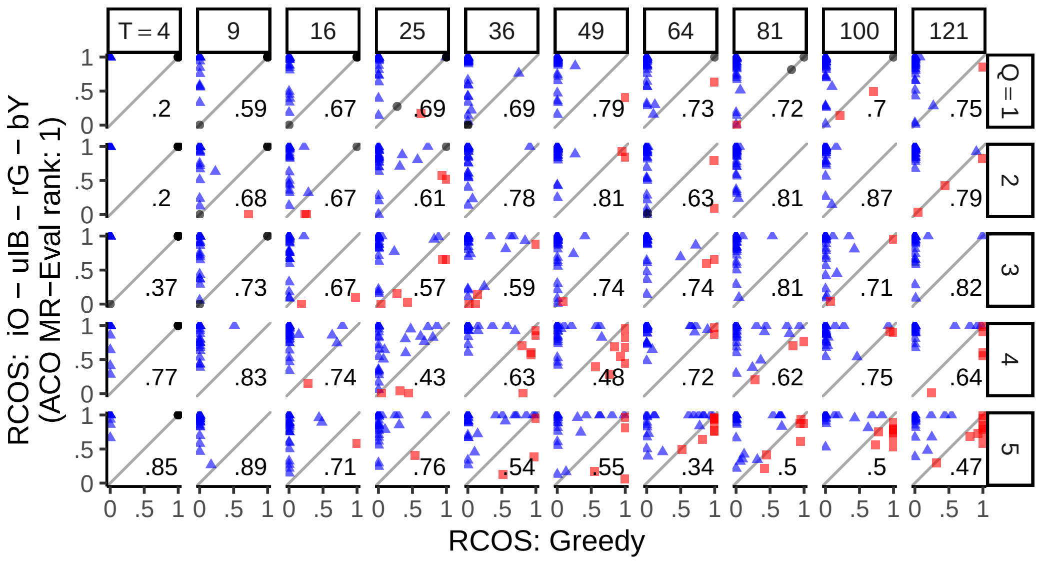

Figure 2: Performance of the best ASPV variant from the M-Eval phase, $RCOS^{\text{eval}}(P^{i,1}_{\text{iO − uIB − rG − bY}}, \mathcal{P}^i)$, against performance of Greedy, $RCOS^{\text{eval}}(P^{i,1}_{\text{Greedy}}, \mathcal{P}^i)$, for the 1,500 new problem instances of the A-Eval phase grouped by number of scans Q (on columns) and horizon T (on rows); the *MSDG* (average of the y-axis value minus x-axis value), rounded to the 2nd decimal, is presented for each group; blue triangles are cases where the ASPV variant solution is best, red squares are cases where the greedy solution is best, and black circles are ties (A-Eval phase)

6.3. Further Evaluation of the Five Best ASPV Variants (A-Eval Phase)

Figure 2 shows the performance of the best ASPV variant from the M-Eval phase, iO – uIB – rG – bY, against that of Greedy on new instances. The mean signed difference between the variant and Greedy, *MSDG*, is displayed for each instance group. We see that ASPV outperforms Greedy in the vast majority of the runs and that the *MSDG* is higher than .5 in favor of the iO – uIB – rG – bY variant on 41 groups out of 50. Each point on the graph corresponds to a single instance among the 1,500 instances in the benchmark. There are a few specific instances where the variant did not improve the solution over Greedy in the allowed time (red squares). In practice, these can be considered as ties since the solution of Greedy is readily available and can be used as the retained solution. As such, there would be no real negative impact to using the ASPV algorithm to search for a better solution in practice. Moreover, for the vast majority of instances ASPV finds a solution that is as good (black circles) or better (blue triangles) than Greedy in the allocated time. Similar conclusions can be drawn for the other top four variants (see the supplementary material).

6.4. Summary of Results

Our results show that ASPV perform best on the M-Eval and A-Eval instances, supporting its generalization potential to other OSPV instances, following a rigorous configuration phase involving multiple variants. In addition, we conclude that restarts, when used in conjunction with a suitable pheromone update scheme such as uIB (iteration-best), uRB (restart-best), or uORBU (on updates of the restart-best solution) are quite efficient for the OSPV. Problem-specific pheromone initialization and boosting also proved useful in our context since most top ASPV variants use these mechanisms. Furthermore, the best ASPV variant provides clearly superior results to the greedy heuristic, which in turn tends to outperform CPLEX on the larger instances.

7. Conclusion

In this paper, we presented the ant search path with visibility (ASPV) algorithm, an ant colony optimization (ACO) algorithm developed specifically for the optimal search path problem with visibility (OSPV), a path planning problem with visibility from search theory. Since the OSPV problem's complexity grows exponentially with the total search time available (T) and combinatorially with the number of scans (Q), our tailored ACO metaheuristic proved useful for obtaining high quality search plans.

In addition to providing a metaheuristic implementation for an active research area in search theory, namely, the optimal search path problem and its variants, we added to the body of empirical evidence in the ACO literature. In fact, we benchmarked a total of 96 algorithm variants, where a variant is a combination of pheromone initialization scheme, pheromone update scheme, diversification mechanism in the form of restarts, and intensification mechanism in the form of a pheromone boosting procedure. Our best variant involved problem-specific pheromone initialization, iteration-best pheromone updates, a geometric restart procedure and pheromone boosting. This variant as well as most variants, outperformed a state-of-the-art general-purpose MILP solver as well as a problem-specific greedy heuristic on our largest and more realistic benchmark instances. This supports the practical usefulness of ACO for path planning problems based on search theory, and adds practical results for detection search problems.

Moreover, our findings are in line with recent results and observations from the ACO literature. First, our best variants, in the OSPV context, use restarts based on a predefined schedule using an increasing sequence of allowed cycles without improvement. It was also observed in the literature on ACO that restarts improve diversification and convergence for other problems (López-Ibáñez et al., 2016). Second, we observed, in the context of the OSPV, a lack of convergence from the less restrictive pheromone updates procedures, e.g., the “all ants” (uAA) updates, and a better convergence for the variants where the “best” solution contributes strongly to the pheromone trails. This is consistent with the observation that more “elitist” ACO implementations tend to have better convergence than more “permissive” ones (Dorigo & Stützle, 2019). Third, in our experiments, the best variants share the following characteristics: a problem-specific pheromone initialization (iO) with boosting (bY), a restart-best update or an iteration-best update procedure (either uRB or uIB) along with restarts (either a Luby or geometric restart schedule, rL or rG). This enables the variants to better balance exploitation (intensification) and exploration (diversification) which is a core characteristic of successful ACO algorithms (Dorigo & Stützle, 2019).

Finally, the OSPV problem formulation and the best ASPV algorithm variants open the door to search pattern optimization. As such, they are milestones in the development of operational decision support systems for search and rescue planning (where search patterns are currently fixed), further improving search operations planning and increasing the potential to save lives.

Acknowledgments

Computer resources were provided in part by *Calcul Québec* (www.calculquebec.ca) and Compute Canada (www.computecanada.ca). Funding: This research was funded by NSERC [grant number RGPIN-2021-03495, DGECR-2021-00189] and by FRQNT [grant number 2017-B3-199085].

References

Abi-Zeid, I., Morin, M., & Nilo, O. (2019). Decision support for planning maritime search and rescue operations in canada. *Proceedings of the 21st International Conference on Enterprise Information Systems*, 328–339.

Audemard, G. & Simon, L. (2012). Refining restarts strategies for SAT and UNSAT. *International Conference on Principles and Practice of Constraint Programming*, 118–126.

Berger, J., Boukhtouta, A., Benmoussa, A., & Kettani, O. (2012). A new mixed-integer linear programming model for rescue path planning in uncertain adversarial environment. *Computers & Operations Research*, 39(12), 3420–3430.

Berger, J. & Lo, N. (2015). An innovative multi-agent search-and-rescue path planning approach. *Computers & Operations Research*, 53, 24–31.

Birattari, M. (2004). On the estimation of the expected performance of a metaheuristic on a class of instances. Technical report, Technical Report TR/IRIDIA/2004-01, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Charnes, A. & Cooper, W. W. (1958). The theory of search: Optimum distribution of search effort. *Management Science*, 5(1), 44–50.

Dai, Q., Ji, J., & Liu, C. (2009). An effective initialization strategy of pheromone for ant colony optimization. *Proceedings of the 4th International Conference on Bio-Inspired Computing*, 398–401.

Ding, Y. F. & Pan, Q. (2011). Path planning for mobile robot search and rescue based on improved ant colony optimization algorithm. *Applied Mechanics and Materials*, volume 66, 1039–1044.

Dorigo, M. & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3), 243–278.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1), 29–41.

Dorigo, M. & Stützle, T. (2019). Ant colony optimization: Overview and recent advances. *Handbook of Metaheuristics*, volume 272, 311–351. Springer International Publishing.

Eagle, J. & Yee, J. (1990). An optimal branch-and-bound procedure for the constrained path, moving target search problem. *Naval Research Logistics*, 38(1), 110–114.

Fang, C. & Anstee, S. (2010). Coverage path planning for harbour seabed surveys using an autonomous underwater vehicle. *OCEANS'10 IEEE SYDNEY*, 1–8.

Foraker, J., Royset, J. O., & Kaminer, I. (2016). Search-trajectory optimization: Part I, formulation and theory. *Journal of Optimization Theory and Applications*, 169(2), 530–549.

Frost, J. R. (1999). Principles of search theory, part I: Detection. *Response, 17 (2)*, 1–7.

Frost, J. R. & Stone, L. D. (2001). Review of search theory: Advances and applications to search and rescue decision support. Technical report, U.S. Department of Transportation, United States Coast Guard.

Goerzen, C., Kong, Z., & Mettler, B. (2010). A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57(1), 65–100.

Grogan, S., Pellerin, R., & Gamache, M. (2018). The use of unmanned aerial vehicles and drones in search and rescue operations – A survey. *Proceedings of the PROLOG.*

Hoos, H. H. & Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications.* Elsevier.

Jovanovic, R., Tuba, M., & Voß, S. (2019). An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research*, 274(1), 78–90. `https://doi.org/10.1016/j.ejor.2018.09.038`

Kratzke, T. M., Stone, L. D., & Frost, J. R. (2010). Search and rescue optimal planning system. *Proceedings of the 13th Conference on Information Fusion (FUSION)*, 1–8.

Lau, H., Huang, S., & Dissanayake, G. (2008). Discounted MEAN bound for the optimal searcher path problem with non-uniform travel times. *European journal of operational research*, 190(2), 383–397.

Lo, N., Berger, J., & Noel, M. (2012). Toward optimizing static target search path planning. *2012 IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 1–7.

Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4), 173–180.

López-Ibáñez, M., Stützle, T., & Dorigo, M. (2016). Ant colony optimization: A component-wise overview. *Handbook of Heuristics*, 311–351. Springer International Publishing.

Minister of National Defence (2013). *Quadrennial Search and Rescue Review.* https://www.publicsafety.gc.ca/cnt/rsrcs/pblctns/archive-nss-qdrnnl-rvw/archive-nss-qdrnnl-rvw-en.pdf. Accessed: 2021-07-23.

Mirjalili, S., Song Dong, J., & Lewis, A. (2020). Ant Colony Optimizer: Theory, Literature Review, and Application in AUV Path Planning. *Nature-Inspired Optimizers: Theories, Literature Reviews and Applications*, Studies in Computational Intelligence, 7–21. Springer International Publishing. `https://doi.org/10.1007/978-3-030-12127-3_2`

Morin, M., Lamontagne, L., Abi-Zeid, I., Lang, P., & Maupin, P. (2009). The optimal searcher path problem with a visibility criterion in discrete time and space. *Proceedings of the 12th International Conference on Information Fusion*, 2217–2224.

Morin, M., Lamontagne, L., Abi-Zeid, I., & Maupin, P. (2010). The ant search algorithm: An ant colony optimization algorithm for the optimal searcher path problem with visibility. *Advances in Artificial Intelligence: 23rd Canadian Conference on Artificial Intelligence*, 196–207.

Morin, M., Papillon, A.-P., Abi-Zeid, I., Laviolette, F., & Quimper, C.-G. (2012). Constraint programming for path planning with uncertainty. *International Conference on Principles and Practice of Constraint Programming*, 988–1003.

Morin, M. & Quimper, C.-G. (2014). The Markov transition constraint. *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 405–421.

Paull, L., Saeedi, S., Seto, M., & Li, H. (2012). Sensor-driven online coverage planning for autonomous underwater vehicles. *IEEE/ASME Transactions on Mechatronics*, 18(6), 1827–1838.

Perez-Carabaza, S., Besada-Portas, E., Lopez-Orozco, J. A., & Jesus, M. (2018). Ant colony optimization for multi-uav minimum time search in uncertain domains. *Applied Soft Computing*, 62, 789–806.

Raap, M., Meyer-Nieberg, S., Pickl, S., & Zsifkovits, M. (2017a). Aerial vehicle search-path optimization: A novel method for emergency operations. *Journal of Optimization Theory and Applications*, 172(3), 965–983.

Raap, M., Preuß, M., & Meyer-Nieberg, S. (2019). Moving target search optimization – A literature review. *Computers & Operations Research*, 105, 132–140.

Raap, M., Zsifkovits, M., & Pickl, S. (2017b). Trajectory optimization under kinematical constraints for moving target search. *Computers & Operations Research*, 88, 324–331.

Richardson, H. R. (2014). Search theory. *Wiley StatsRef: Statistics Reference Online*. American Cancer Society. `https://doi.org/https://doi.org/10.1002/9781118445112.stat00134`

Sato, H. & Royset, J. O. (2010). Path optimization for the resource-constrained searcher. *Naval Research Logistics*, 57(5), 422–440.

Schulte, C., Tack, G., & Lagerkvist, M. (2019). *Modeling and Programming with Gecode*.

Simard, F., Morin, M., Quimper, C.-G., Laviolette, F., & Desharnais, J. (2015). Bounding an optimal search path with a game of cop and robber on graphs. *International Conference on Principles and Practice of Constraint Programming*, 403–418.

Solnon, C. (2002). Boosting ACO with a preprocessing step. *Applications of Evolutionary Computing: EvoWorkshops*, 163–172.

Stewart, T. (1979). Search for a moving target when the searcher motion is restricted. *Computers and Operations Research*, 6, 129–140.

Stone, L. D. (2004). *Theory of Optimal Search*. Academic Press.

Stone, L. D., Royset, J. O., Washburn, A. R., et al. (2016). *Optimal search for moving targets.* Springer.

Tange, O. (2011). GNU parallel – The command-line power tool. *;login: The USENIX Magazine*, 36(1), 42–47. `https://doi.org/http://dx.doi.org/10.5281/zenodo.16303`

Trummel, K. & Weisinger, J. (1986). The complexity of the optimal searcher path problem. *Operations Research*, 34(2), 324–327.

Verbeeck, C., Sörensen, K., Aghezzaf, E.-H., & Vansteenwegen, P. (2014). A fast solution method for the time-dependent orienteering problem. *European Journal of Operational Research*, 236(2), 419–432. `https://doi.org/https://doi.org/10.1016/j.ejor.2013.11.038`

Vermeulen, J. & Van Den Brink, M. (2005). The search for an alerted moving target. *Journal of the Operational Research Society*, 56(5), 514–525.

Walsh, T. (1999). Search in a small world. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1172–1177.

Williams, D. P. (2010). On optimal AUV track-spacing for underwater mine detection. *2010 IEEE International Conference on Robotics and Automation*, 4755–4762.

Wu, H. & van Beek, P. (2007). On Universal Restart Strategies for Backtracking Search. *Principles and Practice of Constraint Programming – CP 2007*, volume 4741, 681–695. Springer Berlin Heidelberg.

Yi, W. & Kumar, A. (2007). Ant colony optimization for disaster relief operations. *Transportation Research Part E: Logistics and Transportation Review*, 43(6), 660–672. `https://doi.org/https://doi.org/10.1016/j.tre.2006.05.004`

Yu, X., Chen, W.-N., Gu, T., Yuan, H., Zhang, H., & Zhang, J. (2019). ACO-A*: Ant Colony Optimization Plus A* for 3-D Traveling in Environments With Dense Obstacles. *IEEE Transactions on Evolutionary Computation*, 23(4), 617–631. `https://doi.org/10.1109/TEVC.2018.2878221`

Yuan, Y. & Wang, D. (2009). Path selection model and algorithm for emergency logistics management. *Computers & industrial engineering*, 56(3), 1081–1094.

Zhu, L., Gong, Y., Xu, Y., & Gu, J. (2019). Emergency relief routing models for injured victims considering equity and priority. *Annals of Operations Research*, 283(1), 1573–1606. `https://doi.org/10.1007/s10479-018-3089-3`