Autonomy Oriented Computing Applied to Image Processing and Feature Extraction

Branko Samarzija
University of Zagreb, Faculty of Electrical Engineering and Computing
Department of Electronics, Microelectronics, Computer and Intelligent Systems
branko.samarzija@fer.hr

Abstract—This paper presents an overview of Autonomy Oriented Computing (AOC) and related approaches to image processing and feature extraction problems. The basic elements of such AOC systems are autonomous entities placed in an environment. The environment, in our case, is viewed as a two-layer 2D lattice containing an image in the 1st layer and a notice board at the 2nd layer. The environment serves as the place where autonomous entities reside, roam and operate. The goal of each autonomous entity, here viewed as a distributed computational agent, is to effectively locate and label image feature pixels, by exhibiting a number of reactive and rational behaviors, such as diffusion, breeding and communication. The entity behavioral repository related to interaction with the image, consists of operations such as testing whether certain pixel color intensity falls into a specific range of values or if the observed pixel match with the specified criteria of the regions homogeneity. It also have operations such as a convolution with the specified mask or evaluation of the array of the pixel values which entity has memorized during it’s search through the image. Each autonomous entity while selecting which behavior to execute is directed by its state, behavior functions, and/or interaction with other entities. Similar techniques have been performed with related optimization algorithms based on artificial ant colonies (Ant Colony Optimization (ACO)) and Particle Swarm Optimization (PSO). Autonomy Oriented Computing can be seen as a more general form of the PSO and ACO algorithms, where agent entities are defined generally with behavioral traits not only directed to optimization, but to any parallel application which can be performed through the interaction of many loosely coupled entities.

Keywords: Autonomy Oriented Computing, Ant Colony Optimization, Particle Swarm Optimization, Swarm Intelligence, Image processing, Feature extraction

I. INTRODUCTION

Nature has always been the primary source of inspiration for modeling new approaches in computation. Observations made on some natural phenomena give us clues what useful characteristics should our computational approach consist of and present a good basis upon artificial model, that performs some specific task, can be developed. One of the earliest examples presents Turing test, which was and still is a test of a machine’s ability to demonstrate intelligence, or John von Neumann’s model of automaton with self-reproducing ability [1]. Since then, many disciplines that have their roots in those early experiments have been created, but divide between computational scientists has grown also. One side takes path to create faster, more efficient algorithms and hardware that exhibit centralized control, while others place less emphasis on speed and efficiency than on robustness, adaptability and emergent organization from interaction of many loosely coupled entities. These later approaches became to be known as biologically inspired computing [2]. Biologically inspired computing is considered as philosophy, rather than a separate field, that links various disciplines such as artificial intelligence, evolutionary computation, bio-robotics, artificial life, and agent-based systems [2]. In this paper, we are especially interested in a relatively new, bottom-up, biologically inspired, multi-entity approach of complex systems modeling and problem solving called Autonomy Oriented Computing (AOC).

A. Introduction to AOC

In many scientific researches or real–world applications that attempt to solve specific problem in a parallel or distributed fashion, use of autonomous agents can provide an efficient way to describe solution process or relation between solution components. Usually, the notion of the autonomous agents is taken in a broader sense, encompassing a wide spectrum of computational systems. In some systems agents have physical embodiment, such as robotic systems where robotic agents efficiently manipulate objects in a 3D Cartesian environment. Other systems presume agents as computationally coded software agents, such as Internet search agents that proactively search for the useful information within highly connected Web of Internet servers, or optimization agents that cooperatively analyze and perform a number of numerical evaluations over a given dataset in order to extract some information or narrow down a large search space to a smaller set of possibilities [3]. No matter what agent purpose is, an agent entity is rarely operating alone. Instead of that, multiple agent entities are usually observed as the part of a Multi-Agent System (MAS). The term multi-agent system refers to all types of software systems composed of multiple semi-autonomous components. MAS considers how a particular task can be solved by a number of modules (agent entities) which cooperate by dividing and sharing the knowledge about the problem and its solution. Research in MAS is focused on the behavior of collections of loosely coupled autonomous agent entities, aimed at solving a given problem. The four main characteristics of an intelligent multi-agent system, which are reflected through the properties and behaviors of its agent entities, are situatedness, autonomy, flexibility and social interaction [4].

The situatedness of an agent entity means that the agent is
physically or biologically situated in the environment from which it receives input and in which it is active and can also effect changes. The part of the environment affected by an agent activity presents a local part of the environment to the corresponding agent. Examples of environments in which agents are situated include the Internet, game play or robotic system. On the other hand, if an optimization task is performed, environment can be imagined as a multi-dimensional problem space, where each location where agent entity is situated (i.e. point) presents the best solution to the optimization function found by the corresponding entity, so far.

**Autonomy** is ability of an agent entity to interact with its environment without direct intervention of other entities. To do this, agent entity must have control over its own actions and internal state. Behaviors (sometimes called behavior patterns [5]) that govern entity actions can be predetermined and applied immediately upon certain condition in agent - environment interaction is achieved. Agents that respond to stimuli received from its environment in an appropriate and timely fashion without reasoning about the alternative solutions are called **responsive or reactive** agents. On the other hand, an agent entity may have an ability to learn from its experience and to improve its performance over time by dynamically adjusting or acquiring behaviors. An agent that responds to situations in its environment not only according to its present stimuli but also searches for alternatives, depending on its current situation, and is able to be goal oriented and opportunistic, is held to be **proactive**. An agent entity that is both reactive and proactive, depending on its current situation, is a **flexible** agent entity [4].

While operating together, in the course of problem solving, agent entities have only a part of the knowledge about the problem to be solved. Thus, in order to solve a problem that is beyond their scope, individual agent entities rely on their communication with other agent entities or with the environment. Actions performed by entities that are related to any kind of communication are called **social interaction**.

The goal of the AOC is modeling and deploying autonomy, where autonomous entities are considered as the smallest and simplest building blocks. Each autonomous entity is characterized by its internal states, evaluation functions, goals, primitive behavior and behavioral rules. The main characteristics of the autonomous entities are [6]:

- **Autonomy**: Entities are reactive and/or proactive individuals that act independently,
- **Emergency**: During the course of the interactions among individuals, entities can exhibit complex behaviors that are not predefined within their initial set of behaviors or behavioral rules,
- **Adaptive**: Entities have the ability to change their behaviors in response to the changes in the environment in which they are situated in order to best fit the problem they are attempting to solve,
- **Self–organization**: Entities are able to organize themselves to achieve the above behaviors.

It can be seen that AOC characteristics emergency are adaptiveness are closely related to the traditional MAS agent properties of flexibility and social interaction. However, the emphasis is on self-organization [6], because it describes how entities are able to organize themselves in order to achieve their goals. This characteristic is usually described in the algorithmic way and it is characteristic for a given problem.

Study of AOC is essentially based on the theory and concepts researched in cellular automata [7], multi–agent systems, artificial life, evolutionary automata [8] and swarm intelligence. However, unlike evolutionary automata and its investigation of deep scientific questions such as: “Under what conditions does evolution produce complexity and diversity?”, it aims to be a generic model or framework for complex systems modeling and problem solving where a special attention is paid to the role of self-organization among individual entities.

Preliminary study of the AOC encompassed a wide spectrum of applications including constraint satisfaction problem solving [9], mathematical programming [10], optimization [11], image processing [12] and data mining [13]. During the last 3 years, most of the research has been maid in AOC application on WWW, mostly including various types of data gathering in Distributed and Dynamic Networks [14] or discovering in global Network Communities [15]. The reasons for such state lie in the fact that AOC-based methods are applicable for solving problems which are distributed and/or decentralized by Nature, or computation in distributed and/or decentralized manner is required due to the lack or unsuitability of central controller. Problems related to the distributed networks, such as peer-to-peer (P2P) networks, usually comply with those conditions. Another important property of the AOC-based methods is their suitability for mining dynamically changing or evolving data rather than dealing with the static ones. That is, once when the system finds suitable solution for a given problem, it will respond much faster to the changes compared to other systems/algorithms that may need to start computation all over again.

In this work, we address problems related to low-level image processing and feature detection such as color-based image segmentation, object contour detection, Euler number computation and simple region classification according to their previously detected features (color, size, and Euler number) by presenting a new AOC-based system that successfully finds solutions to each of those problems. The following sections provide the formal definition of the AOC, as well as an overview of the related disciplines that are, namely: cellular systems, ant colony optimization and particle swarm optimization. A special interest is directed to image processing tasks that have been tackled by any of those disciplines. The experimental part shows the results of the proposed system. The work concludes with the discussion about the benefits of the proposed AOC-based system application on image processing.
B. Formal definition of the AOC

An AOC system \( \textbf{A} \) is formally defined as the 3 - tuple: \( \textbf{A} = (\mathcal{E}, \mathcal{E}, \Phi) \), where \( \mathcal{E} \) is a set of entity classes containing autonomous entities, \( \mathcal{E} \) is an environment in which entities reside and \( \Phi \) is the system objective function, which is usually a non-linear function of the entity states. The system objective function is incorporated into the performance of the entities through their behavioral rules by implicitly regulating entities toward the desired configuration [6].

The set \( \mathcal{E} \) is defined as: \( \mathcal{E} = \{ e_1, e_2, \ldots, e_N \} \), in which each element \( e_i \in \mathcal{E} \), \( i = 1, 2, \ldots, N \), represents one entity class and \( N \) is the number of entity classes in \( \mathcal{E} \). Element \( e_i \in \mathcal{E} \) is defined as a multiset \( e_i = (\{ e_{i1}, e_{i2}, \ldots, e_{iN} \}) \), where \( e_{ij} \) is the only element of \( e_i \) and \( \mu_G : e_i \rightarrow \mathbb{N} \) is a function from \( e_i \) to the set of natural numbers \( \mathbb{N} \), that measures a number of occurrences (i.e. multiplicity) of \( e_{ij} \) in \( e_i \).

A is a dynamical system where the number of entity classes as well as the number of entities in each class is changing in discrete time steps. The number of entity classes at discrete time moment \( t = 0, 1, 2, \ldots \) is equal to the cardinal number of \( \mathcal{E} \) at time \( t \) and is denoted as \( \text{card}(\mathcal{E}(t)) \). (\( \mathcal{E}(t) \) denotes set of entity classes at the moment \( t \)). Total number of entities in \( \mathcal{E}(t) \) is the sum of all multiplicities of entities contained in all entity classes in \( \mathcal{E}(t) \). It is equal to \( N(\mathcal{E}(t)) = \sum_{i=1}^{\text{card}(\mathcal{E}(t))} \mu_E(e_i) \).

**Example 1.** Consider the following example:

\[ \mathcal{E}(0) = \{ e_1, e_2, e_3, e_4, e_5 \}, \quad e_1 = \{ e_11, e_12, e_13 \}, \quad e_2 = \{ e_21, e_22 \}, \quad e_3 = \{ e_31, e_32, e_33 \}, \quad e_4 = \{ e_41, e_42, e_43, e_44 \}, \quad e_5 = \{ e_51, e_52, e_53 \}. \]

\( \mathcal{E}(0) \) presents a set of 4 entity classes at the moment \( t = 0 \). The multiplicities of entities in each class are: \( \mu_E(e_1) = 3 \), \( \mu_E(e_2) = 2 \), \( \mu_E(e_3) = 1 \), \( \mu_E(e_4) = 5 \). Total number of entities in \( \mathcal{E}(0) \) is computed as \( N(\mathcal{E}(0)) = \sum_{i=1}^{\text{card}(\mathcal{E}(0))} \mu_E(e_i) = 3 + 2 + 1 + 5 = 11 \).

Suppose that during the moment \( t = 0 \) we create a new class of entities called \( e_6 \) containing 2 entities, and also remove 3 entities from the class \( e_4 \), and add 2 entities to the class \( e_3 \). Set of entities at the moment \( t = 1 \) will look like: \( \mathcal{E}(1) = \{ e_1, e_2, e_3, e_4, e_5, e_6 \} \), were \( e_1 = \{ e_11, e_12, e_13 \} \), \( e_2 = \{ e_21, e_22 \} \), \( e_3 = \{ e_31, e_32, e_33 \} \), \( e_4 = \{ e_41, e_42, e_43, e_44 \} \), and \( e_5 = \{ e_51, e_52, e_53 \} \). Total number of entities in \( \mathcal{E} \) at the moment \( t = 1 \) is \( N(\mathcal{E}(1)) = \sum_{i=1}^{\text{card}(\mathcal{E}(1))} \mu_E(e_i) = 3 + 2 + 3 + 2 + 2 + 2 = 12 \).

An autonomous entity \( e \) is defined as a 5 - tuple \( \langle S, F, G, B, R \rangle \), where \( S \) describes entity’s current state, \( F \) is the entity’s evaluation function, \( G \) is a set of entity’s goals, \( B \) and \( R \) define sets of the entity’s primitive behaviors and behavioral rules, respectively. Based on the differences in \( S, F, G, B \) and \( R \), entities in an AOC system are categorized into different classes. System that contains entities that belong to different classes has heterogeneous design.

State \( S \) of autonomous entity \( e \) is characterized by an n-tuple of static or dynamical descriptors (parameters):

\[ S = (s_1, s_2, \ldots, s_{N_d}) \tag{1} \]

There are numerous ways to assign meanings to each static or dynamical parameter of \( S \) since every assignment depends on a given problem. However, the most common parameters of the entity’s state are defined as follows (having in mind that the AOC system is intended to solve image processing problems):

\[ S = (\text{internal\_state}, \text{age}, p, m, L_s), \tag{2} \]

where: \text{internal\_state} defines entity’s internal state which can be: active is the initial state in which the entity stays while roaming from pixel to pixel and performing pixel evaluation; communicable is the state that the entity enters when it finds image feature and is also surrounded by other entities, so according to it’s behaviours, it wants to share information with other entities directly; sleep is the state that the entity enters when it finds image feature, but still expects to be utilized in further algorithm steps (it can be awaken to the active state again); dead is the state that entity enters when its age exceeds its lifespan. Such an entity is not utilizable for further image processing, so it is removed from the environment. The parameter age presents entity’s ages. All entities initially placed in the system or created as a result of performing self-reproduction behavior from its parent have initial ages set to zero. The entity’s age is incremented during the performance of some types of behaviors from the \( B \) (usually diffusion (that is moving of the entity from pixel to another)), so through this parameter maximal range an entity can reach is controlled. When entity reaches certain ages (greater than \( L_s \)), it is removed from the system (entity enters state dead prior to be removed). Parameter \( p \) is entity’s position in the environment. If environment contains image, this parameter corresponds to the pixel coordinates. Parameter \( m \) is entity’s memory. In it’s memory, entity can write necessary data during it’s image exploration (e.g. minimal or maximal pixel values, distances between detected features or the history about it’s path from one pixel to another).

Before an entity \( e \) fires its behavioral rules to select its primitive behavior, it assesses its current condition (i.e. state of the environment at its current position, its own internal\_state and/or those of its neighbors). Formally, evaluation function \( F \) is defined as follows:

\[ F : D_s \rightarrow R, \tag{3} \]

where \( D_s \) is application dependent set of values (usually it corresponds to the position \( p \) of the autonomous entity within the environment), but it can also be the state of the environment or the entity. \( R \) is the range of function \( F \) (usually set of real or integer values).

**Example 2.** Observe the example in which entity \( e \) is positioned at the pixel location with coordinates \( p = (i, j) \) (as depicted in Fig. 1). Entity has a goal to evaluate gradient value in \( x \) and \( y \) direction. To do that, entity performs convolution with the Sobel masks \( d_x = [v, 0, -v] \), \( d_y = d_x^T \), where \( v = [1, 2, 1]^T \), \( 0 = [0, 0, 0]^T \). If the result is greater for \( d_x \), entity will change its current position pixel intensity value to 0 and step one pixel north, otherwise entity will step one pixel south (Fig. 1.b).
The neighbors of entity $e$ in the given AOC system, are a group of entities $L^e$ that satisfy neighborhood criterion, which is an application-dependent constraint (e.g., distance between entities).

**Example 3.** Consider the example in which 10 entities $e_1, e_2, \ldots, e_{10}$ are placed onto a 2-dimensional image (see Figure 2). We suppose that each entity occupies only one image pixel at a time and that each image pixel can be occupied by only one entity. If we define entity neighborhood as being 4-connected, having a similar meaning as the pixel neighborhoods in image processing, then the neighbors of the entity $e$ are entities $e_2, e_4, e_6, e_8$ (fig 2.a). If entity neighborhood is 8-connected than, the neighbors of the entity $e_1$ will be all the entities, except entity $e_{10}$ (fig 2.b). If we express distance in two dimensions by Minkowski metric $L_k(e_1, e_i) = (|p(e_1).x - p(e_i).x|^k + |p(e_1).y - p(e_i).y|^k)^{\frac{1}{k}}$, where $p(e_i).x$ and $p(e_i).y$, $i = 1, 2, \ldots, 10$ corresponds to the pixel coordinates in image lattice at which entity $e_i$ is placed, than it is said that all the entities $e_2, e_3, \ldots, e_{9}$ (except $e_{10}$) are in the neighborhood of the entity $e_1$ with radius $= 1$ using the $L^e$ norm (According to the $L^e_{\infty}$ norm, distance between two points in d-dimensional space corresponds to the maximum of distances between the projections of those two points onto each of d coordinate axes. For all entities $e_2, e_3, \ldots, e_{9}$, this value is equal to 1). All the entities $e_2, e_3, \ldots, e_{10}$ are in the neighborhood of entity $e_1$ with radius $= 2$ (fig 2.c). Besides $L^e_{\infty}$ norm, other norms such as $L^e_1$, $L^e_2$ can be used for expressing neighborhood distances.

An entity $e$ can perform a set of primitive behaviors $B = \{b_1, b_2, \ldots, b_{|B|}\}$ where $|B|$ denotes the number of primitive behaviors. Each primitive behavior is a mapping in one of the following forms:

- **Breeding:**

  $$b_i : \mathcal{E}^\infty \rightarrow \mathcal{E}^\infty \cup \{e_j, e_j, \ldots, e_j\} \quad (4)$$

  where $m \in \mathbb{N}$. By performing this behavior, the entity $e_j \in \mathcal{E}^\infty$ extends the original multi-set $\mathcal{E}^\infty$ by the $m$ new offspring entities ($e_j$, all the offspring entities have the same indexes). It is problem-dependent to decide what information the offspring entities inherit from its parent. The offspring entities have to be in the same class as their parent. If the offspring entities have age in their state $S$, it is always initially set to zero.

  - **Self-multiplication (i.e. Cloning):**

    $$b_i : \mathcal{E}^\infty \rightarrow \mathcal{E}^\infty \cup \{e_j, e_j, \ldots, e_j\} = \mathcal{E}_{\text{New}}^\infty \quad (5)$$

    where $m \in \mathbb{N}$. By cloning, the number of occurrences of the entity $e_j$ in the multi-set $\mathcal{E}^\infty$ is increasing by $m$.

    That is $\mu_{\mathcal{E}_{\text{New}}^\infty}(e_j) = \mu_{\mathcal{E}^\infty}(e_j) + m$. Cloning is defined similarly to breeding. However, cloning supposes that the newly added entities will be the exact copies of its parent (parameters like internal state, memory, age will be identical), while by breeding such parameters will be set to initial values that are predetermined like ages (initially set to zero), or derived from its parent attributes like search/diffusion direction (e.g. parent entity while searching for the image feature concluded that feature is not located in the upper left region of the image, so it directs its offspring entities elsewhere).

- **Die:**

  $$b_i : \mathcal{E}^\infty \rightarrow \mathcal{E}^\infty_{\text{New}} \quad (6)$$

  where $\mathcal{E}^\infty_{\text{New}} = \mathcal{E}^\infty_{\text{New}} \setminus \{e_j\}$. When applies Die behavior, the entity $e_j$ vanishes from the environment, which results with $\mu_{\mathcal{E}^\infty_{\text{New}}}(e_j) = \mu_{\mathcal{E}^\infty}(e_j) - 1$. This behavior is only possible if $\mu_{\mathcal{E}^\infty}(e_j) > 0$. But however, since entity $e_j$ must exist in the environment to be able to perform any kind of behavior, this condition is always satisfied.

- **Change internal state:**

  $$b_i : D_S \rightarrow D_S \quad (7)$$

  $D_S$ denotes entity state space.

- **Change internal state and those of neighbors:**

  $$b_i : D_S \times \prod_{e_i \in \mathcal{E}} D_{E(e_i)} \rightarrow D_S \quad (8)$$

  $D_S$ denotes entity state space.

Environment $\mathcal{E}$ is characterized by a set $\mathcal{E}S = \{e_{s1}, e_{s2}, \ldots, e_{s_{|s|}}, e_{s_{|s|}}\}$, where each $e_{si} \in D_{e_{si}}$ describes one statical or dynamical attribute of $\mathcal{E}$, $D_{e_{si}}$ is a set of possible values of $e_{si}$, and $|\mathcal{E}S|$ is the number of
attributes of $E$. At each moment, $t = 0, 1, 2, \ldots$, state of $E$ is denoted by $E^{S(t)}$. The state space of $E$ is $D_{ES} = D_{es1} \times D_{es2} \times \cdots \times D_{es1} \times \cdots \times D_{esNes}$.

**Example 4.** Consider this simple constraint satisfaction problem (CSP). Assume that environment $ES$ consists of 3 attributes $ES = \{es1, es2, es3\}$. Possible values for each attribute are given as follows: $D_{es1} = \{1, 2, 3, 4, 5, 6\}$, $D_{es2} = \{1, 2, 3, 4\}$, $D_{es3} = \{1, 2, 3, 4, 5\}$. Set of constraints is given as $C = \{es1 \neq es2, es1 > es3\}$. Our goal is to find at least one solution $S(es1, es2, es3)$ that satisfies given constraints. In order to find solution $S$, possible values for each attribute are organized into rows and in each row only one entity is placed (Fig. 3). Each entity first tries to find a solution for the attribute assigned to it by moving horizontally, and after it finds it, it communicates vertically with other entities to prove if mutual constraints have been satisfied. After several iterations entities produce one possible solution $S = \{4, 2, 1\}$.

The environment $E$ serves as a domain in which the autonomous entities roam and operate. For the image processing problems, environment $E$ is characterized as two-layer two-dimensional lattice containing image in the first layer and containing a notice board in the second layer Fig. 4. The notice board is a 2D lattice of the same dimensions as the image in the first layer. Every cell of the notice board contains information related to the pixel at the same coordinates. This information is left and modified by the entities, and it serves as a processing data for every entity that visits corresponding pixel. In other words, an indirect communication among entities is achieved this way.

The system objective function $\Phi$ is a global measurement for the performance of an AOC system. It guides the system to evolve towards certain desired states or patterns. It is defined as a function of states of all entities in the system:

$$\Phi \cdot \sum_{e_i \in E} D_{e_i} \rightarrow R,$$

where $D_{e_i}$ denotes entity’s state space and $R$ is a set of real or integer values.

**Example 5.** Consider the example in which we are monitoring internal states of a set of entities. We expect that algorithm termination condition is satisfied when 3 entities have their internal state set to passive. At the moment $t$ we have 5 entities: $e_1(\text{active})$, $e_2(\text{dead})$, $e_3(\text{passive})$, $e_4(\text{active})$, $e_5(\text{active})$. We calculate $\Phi(t)(\text{active}) = 3$, $\Phi(t)(\text{passive}) = 1$, $\Phi(t)(\text{dead}) = 1$. We conclude that terminating condition is not reached since the number of entities in the passive state is equal to 1. During the time $t+1$ we have 4 entities: $e_1(\text{active})$, $e_3(\text{passive})$, $e_4(\text{active})$, $e_5(\text{passive})$. We calculate $\Phi(t+1)(\text{active}) = 1$, $\Phi(t+1)(\text{passive}) = 3$. Since, we now have 3 entities in the state passive, we conclude that the system has reached termination condition and it stops processing.

II. RELATED DISCIPLINES

In this paragraph we provide an overview of the AOC related disciplines, that were invented prior to AOC theory and that had strong influence on the initial AOC development. Those are: cellular systems and cellular automata (CA), Swarm Intelligence (SI) in general with Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). First we provide an illustrating example of the cellular systems and a short overview of cellular automata (CA) since AOC-based systems, especially those applied to low-level image processing tasks, are based on the ideas derived from the CA theory.

A. Cellular Systems

One of fundamental works on cellular systems that investigated spatial effects in social dynamics is described in [16]. Described model of the cellular system contained a population of agents belonging to two distinct classes distributed on a two-dimensional cellular space, with some cells being left empty. The agents represented two groups of individuals and the cellular space represented their environment. According to the model, agents used a numerical value called attitude to express their stance toward the agents that belonged to their own class and their stance to the agents that belonged to the other class.
During each simulation time step $k$th agent computed value $v^{(k)}_j$ for all empty cells within its $3 \times 3$ neighborhood, including its current position, as:

$$v^{(k)}_j = \sum_{i,j \neq i} \frac{a^{(k)}_{ij}}{((x_i - x_j)^2 + (y_i - y_j)^2)^k},$$  \hspace{1cm} (10)

where $i$ is an index of the analyzed cell ($x_i$ and $y_i$ are its coordinates in cellular environment), $j$ are indexes of all other cells in the cellular space that contain agent (i.e., cell coordinates of all agents in the population), and $a^{(k)}_{ij}$ is the attitude of the $k$th agent toward the agent at cell $j$. If there were empty cells in $k$th agent neighborhood that had value $v^{(k)}_j$ greater than the value assigned to the agent’s current position, agent would move to the position having the highest value. Otherwise, agent would extend its evaluation process to the $5 \times 5$ neighborhood. If no position with higher value have been found in extended neighborhood, agent would stay in its current position until next time step, when the evaluation process repeats.

The research explored several scenarios based on the value of attitude parameter $a_{ij}$, in one scenario, that was called “suspicion”, agent $i$ used two values for $a_{ij}$, if agent $j$ belonged to the same class as agent $i$, then $a_{ij} = a_{own} = 0$, and if agent $j$ belonged to the other class value was $a_{ij} = a_{other} = -1$. Simulation showed that for those parameter agent will form clusters in the cellular space that are very separated. Agent belonging to the other class will be pushed away, while consequently agents belonging to the same class will end up together (fig. 5a - 5c)). However, since agents have neutral attitude towards their own class, there will be more than one cluster of agent belonging to the same class.

In other scenario, called “segregation”, agents had positive attitude towards their own class $a_{ij} = a_{own} = 1$, and negative to the other class $a_{ij} = a_{other} = -1$. Simulation, produced clusters of agents similarly as in the first scenario, but now agents belonging to the same class tend to group together, not just move away from the agents belonging to the other class (fig. 5d-5f)).

### B. Cellular automata

The cellular automata consists of a regular grid of cells where each cell is one cellular automaton. By definition, cellular automaton or simply cell, derives from mathematical concept of automaton, which is a discrete-time system with finite set of inputs $I$, a finite set of states $S$, a finite set of outputs $O$, a state transition function $\phi$ which gives the state at the next time step as a function of the current state and inputs, and an output function $\eta$ which gives the current output as a function of the current state [17]. The state of a cell is determined by the current states of all cells in a surrounding neighborhood (i.e., inputs), and it can be expressed as: $s_i(t + 1) = \phi(s_j(t) : j \in N_i)$, where $s_i(t + 1)$ is the state of automaton $i$ at time $t + 1$, $s_j(t)$ are states of automata in the neighborhood of automaton $i$ (denoted as $N_i$) at time $t$ ($N_i$ also includes current state of automation $i$, $s_i$), and $\phi$ is the state transition function. It can be noted that inputs to the state transition function are only states of the cells, not any other data. Usually, the state transition function (which is equal for each cell), is represented in a form of a transition table (also called rule table), that is, a table which specifies the next state of the cell for every possible configuration of the cell states in its neighborhood. Building such a table becomes impractical for even small number of cell states and small number of cell neighbours. If each cell can be in $k$ states and have $n$ neighbours, then the transition table will have $k^n$ entries (this is a number of the possible state configurations in the neighbourhood), while the number of possible transition tables is $k^{kn}$ (for each state configuration we choose between $k$ possible states). For example, if $k = 2$ and $n = 9$, then the table has $2^9 = 512$ entries, while the number of possible tables is $2^{512}$. The most common type of CA grid is 2D grid, and the most common types of neighbourhoods used by CA are von Neumann neighborhood (that is 4-connected pixel neighbourhood in image processing) and Moore neighborhood (that is 8-connected pixel neighbourhood) (see Fig. 6).

The theory of CA was originally proposed by John Von Neuman [1] and Ulam [18] with the purpose of exploring models of biological self-reproduction. Those early efforts
TABLE I: CA state transitions for the rules #30 and #110

<table>
<thead>
<tr>
<th>current state</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>next state (#30)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>next state (#110)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The two of the most popular examples of CA are the one-dimensional elementary CA (explored by Wolfram) and the two-dimensional game of life (known as the Conway's game of life) [17]. The one-dimensional elementary CA are binary CA aligned in one row where each cell have two neighbors (neighborhood radius is \( r = 1 \)). There are 256 such automata (each cell changes its state according to the states of its neighbors and its own that is \( 2^8 = 8 \) and for the next state it can choose between the two states, which results in \( 2^8 = 256 \) possible transition tables). Wolfram CA that are produced by the rules 30 and 110 (see table I), are shown on Figure 7.

One of the very interesting two-dimensional CA proposed by Chris Langton [17] is shown at Fig.8. This cellular automaton consists of square grid of white/black squares (cells) as the environment and an ant that operates on those cells (in this particular example the operating entity is rather called ant than cellular automaton, and it is named after its inventor: Langton’s ant). The plane in which ant operates is divided into squares (we can imagine it as an image where the squares are image pixels). Each square can be black or white. The ant (imagined as an entity that steps from pixel to pixel) is moving in 4 possible directions at each step it takes. The simple rules the ant applies during its movements are: (1) if the ant is at the white square, turn right 90°, change its color to black, and move forward one unit; (2) if the ant is at the black pixel, turn left 90°, change its color to white, and move forward one unit.

By applying those simple rules, the ant is seemingly behaving stochastically, moving in a random manner, but after around 11000 steps an interesting pattern starts to emerge. It looks like an ant highway, and continues infinitely.

The CA are considered as a suitable tool for image processing due to their local nature and simple parallel computing implementation. The local nature make CA applicable for image operations that can be computed directly on the part of the image without any previous analysis of the whole image (such as image sharpening or smoothing, image noise reduction, or local segmentation), while parallel implementation in general speeds up processing time by simultaneously processing different parts of the image.

Some of the image processing tasks CA have been applied to include: calculating distances to features [22], calculating properties of binary regions such as area, perimeter and convexity [23], performing image enhancement operations such as noise filtering and sharpening [24], detecting edges [25], performing simple object recognition [26].

C. Swarm intelligence

Swarm intelligence (SI) is an artificial intelligence technique involving the study of collective behavior in decentralized systems [27]. Such systems are made up by a population of simple individuals interacting locally with one another and with their environment. Although there is typically no centralized control dictating the behavior of the individuals, local interactions among the individuals often cause a global pattern to emerge. Examples of systems like this can be found in large quantities in the Nature, including ant colonies, bird flocking, animal herding, honey bees, bacteria, and many more. SI refers to the problem-solving behavior that emerges from the interaction between individuals in such systems, and computational SI refers to algorithmic models of such behaviors. These algorithms have shown to be adaptive and
robust in changing environments. As traditional algorithms, which emphasize on centralization, became increasingly inadequate in handling today’s problem, SI algorithms offer an alternative to problem-solving. The two most popular SI paradigms, which have shown the significant research interest in the last decade, are ant colony optimization (ACO) and particle swarm optimization (PSO).

1) Ant Colony Optimization: The Ant Colony Optimization (ACO) is a multi-agent approach to difficult combinatorial optimization problems solved by the observations of real ant colonies. The first ant algorithm, called Ant System (AS), was proposed by Marco Dorigo in [28]. AS is the result of the research on computational intelligence approaches to combinatorial optimization that Dorigo and colleagues applied to the traveling salesman problem (TSP) and to the quadratic assignment problem (QAP). Later on, based on the good performance of the AS obtained on TSP and QAP, Dorigo and colleagues defined the general procedure of constructing algorithms based on the ant concepts called ACO meta-heuristic [27]. As a part of it, besides the original AS, Dorigo and colleagues have proposed modifications to the AS in the form of new algorithms called Ant Colony System (ACS) [29] and MAX-MIN Ant System (MMAS) [30]. ACO meta-heuristic has been applied to many static combinatorial problems (beside TSP and QAP), such as: vehicle routing [31], sequential ordering [32], job-shop scheduling [33], graph coloring [34], constraint satisfaction [35]. The ACO has also been successfully applied to dynamic optimization problems, mostly related to telecommunication networks [36] or data mining [37].

Definition 1. (Definitions from the book Dorigo, Stutzle [33].) Heuristic refers to experience-based techniques for problem solving, learning, and discovery. Heuristic methods are used to speed up the process of finding a satisfactory solution, where an exhaustive search is impractical. A meta-heuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. A meta-heuristic can be seen as a general-purpose heuristic method designed to guide an underlying problem-specific heuristic (e.g. local search algorithm or a construction heuristic) toward promising regions of the search space containing high-quality solutions. A meta-heuristic is therefore a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem.

Recently, it has grown an interest to apply ACO meta-heuristic in image processing and pattern recognition problems like: image segmentation [38], image feature selection [39], edge detection [40], texture segmentation [41] [42], 3-D object segmentation [43]. In the following section we provide a description of the computational model of ant algorithms based on the Ant System algorithm (AS). After that, a more concise example, where the ant algorithm is applied to image processing problem of edge detection is described.

The first observation of the real ant colonies is that each ant colony consists of finite number of ants. This number is constant during short periods of time, but however it can change due to numerous biological factors. In the computational model, the number of ants in the colony is usually constant and equal to N. However, in some papers it has been reported to be changeable through included ant reproduction and cease to exist ability achieved by the aging [38]. The main mechanism by which real ants as well as artificial ants achieve communication and exploration gains is deployment of the chemical substance called pheromone. It is not visible, but ants have strong senses towards its presence and concentration levels. Pheromone trails left by the real ants as they walk over the land or by artificial ants as they explore problem solution space (pheromone trails are usually recorded as a matrix containing real values) serves as a guidance toward shorter paths or better solutions. In the algorithm described next, it will be shown how pheromone values are analysed, increased and decreased. The main characteristics of the artificial ant in the computational model that are considered as different to the properties of the real ants are [28]: memory (for storing information related to the computational problem), artificial ant is not completely blind (for being able to read data from the part of the problem space near it’s current position), discrete time environment (ant algorithms are iterative algorithms where all computational activities are synchronised by iteration step). All those abilities can be extended furthermore, depending on the computational problem. AS algorithm, as proposed by Dorigo [28] is shown at Alg. 1.

Algorithm 1 Ant system (AS) algorithm for TSP

<table>
<thead>
<tr>
<th>Procedure ACO for TSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Initialize data</td>
</tr>
<tr>
<td>2: while not termination criterion satisfied do</td>
</tr>
<tr>
<td>3: Construct Solutions</td>
</tr>
<tr>
<td>4: Local Search</td>
</tr>
<tr>
<td>5: Update Pheromone Trails</td>
</tr>
<tr>
<td>6: end while</td>
</tr>
<tr>
<td>7: Display Results</td>
</tr>
</tbody>
</table>

The AS algorithm was initially explained at the TSP problem. The TSP problem is Np-hard computational optimization problem which is defined as following: For a given list of towns and list of the distances between them, find a shortest possible tour that is going through all towns exactly once. Combinatorial solution for this problem if we observe G towns, has C!/2G solutions. It is obvious, that for even small values of G, this way of solving the problem is unfeasible. The AS algorithms, as it belongs to the group of heuristic algorithms, is able to find solutions to the TSP problem in a relatively short time for big values of G. The procedure 2 describes computational steps in the algorithm initialization phase. It initialises all the necessary data as pheromone map (sets initial pheromone values), compute distances between each town, set initial ants locations (each town can hold one
The pheromone $\tau_{ij}$, associated with the edge joining cities $i$ and $j$, is updated as follows [44]:

$$\tau_{ij}^{(t+1)} = (1 - \rho)\tau_{ij}^{(t)} + \sum_{k=1}^{m} \Delta \tau_{ij}(k), \quad (11)$$

where $\rho$ is the pheromone evaporation rate, $m$ is the number of ants, and $\Delta \tau_{ij}(k)$ is the quantity of pheromone laid on edge $(i, j)$ by ant $k$ which is computed as:

$$\Delta \tau_{ij}(k) = \begin{cases} Q/L_k & \text{if ant } k \text{ used edge } (i, j) \\ 0 & \text{otherwise} \end{cases}, \quad (12)$$

where $Q$ is a constant, and $L_k$ is the length of the path crossed by ant $k$. This equation means that, ants that have found shorter path (better solution) will lay more pheromone on that path, that in turn will attract more ants, and colony will converge to the better solutions. When ant $k$ is located on the node $i$ and has so far visited all the nodes from the set $s^k$, the probability of going to node $j$ is given by:

$$p_{ij}(k) = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{j \in s^k} \tau_{ij}^\alpha \eta_{ij}^\beta} & \text{if node } j \notin s^k \\ 0 & \text{otherwise} \end{cases}, \quad (13)$$

where $\eta_{ij}$ is heuristic information, which is problem dependent. For the TSP problem where the goal is to find the shortest path in the problem graph it can be computed as:

$$\eta_{ij} = \frac{1}{d_{ij}}, \quad (14)$$

where $d_{ij}$ is the distance between nodes $i$ and $j$. We can notice that nodes that are closer to each other will receive greater amounts of pheromone, and therefore more likely became parts of the shortest path. The parameters $\alpha$ and $\beta$ represent influences of the pheromone and heuristic information, respectively. Usually, for the image processing tasks, heuristic information tends to be much more important since this information is obtained from the image content. At the same time, pheromone information is error prone due to ant’s inability to make right decision at the certain pixel position, or due to the inherent randomization effects (i.e. one ant makes random (wrong) step, other ants follow, putting stress on the bad (noisy) image features). That leads to smaller values of $\alpha$, typically $\alpha$ is around 1, while $\beta$ is around 2.5 (assuming that the values of $\tau$ and $\eta$ are both scaled to $[0, 1]$).

Heuristic information can suffer from: noisy image, heuristic function operating on a region that is too small ($3 \times 3$ or $5 \times 5$), observed part of the image is too blurred, heuristic function is unable to extract useful information or extracted information is vague. The heuristic information is the most important function for each ACO related problem. It deals with the problem data directly, while all other computations are the same for each ACO related problem (differs eventually on different parameters values for $\alpha$, $\beta$, $\rho$, $\tau_0$, $N$, $Q$ or algorithm stopping conditions). In the next, we are going to describe how image processing problems were addressed by ACO.

To the best of our knowledge, the idea of applying artificial ant colonies for the image processing tasks was first described in [45]. The idea comes from the study of patterns that could emerge from the interaction of many simple local rules. Artificial ants organized into a swarm were placed at the digital image as their environment in order to perceive it. Pheromones fields that evolved as a result of colony evolution pointed out that, because of their different appearance for the different input images, are suitable for image recognition. Initial conditions of pheromone distribution seemed to have little importance.

This model of an artificial ant colony system was later extended in [38], by adding a self-regulated variation of population size. It was achieved through the concept of ants’ aging. Ants that were successful according to their fitness function were able to reproduce, while the older ants had higher chance to die. The model was successfully applied to image Watershed segmentation enhancement.

ACO edge detection has been proposed in [46]. In that approach, digital image was represented as a perceptual graph, where nodes were image pixels, while connections between the nodes were weighted arcs. The arcs presented relationships among neighboring pixels where each node was connected to its 4 neighbors using image 4-connectivity. According to the applied ACS algorithm, arc weights corresponded to pheromone values, and ants while roaming through the image guided by pixel to pixel intensity difference, adjusted arc weights. In the end, resulting edge image was extracted from the graph using one of 3 policies at each node: maximum node’s arc weight, length of all weights and variance of the weights.

In [40] edge detection using AS algorithm used two dimensional graph representation of the given image for storing information about pheromone trails. In the graph, each node was connected to its 8-neighbors using image 8-connectivity. Graph nodes stored pheromone intensity information related to the corresponding image pixels while arcs were not used.

A number of artificial ants, that were initially distributed randomly over the image graph, traversed image graph and applied Roberts convolution mask of size $3 \times 3$ to determine heuristic information, there called visibility:
The edge image is obtained from the pheromone map values by performing thinning and thresholding operations.

Fig. 9: Edge detection by ACO algorithm as proposed in [40]: The edge image is obtained from the pheromone map values by performing thinning and thresholding operations.

\[ \eta_{i,j} = \frac{1}{I_{\text{Max}}} \max \{|I(i-1,j-1) - I(i+1,j+1)|, |I(i-1,j+1) - I(i+1,j-1)|, |I(i,j-1) - I(i,j+1)|, |I(i,j) - I(i+1,j+1)| \} \]  

\[(15)\]

\((i,j)\) represents pixel (node) coordinates for which visibility is computed \((I(i,j)\) is pixel value, i.e. gray-scale pixel value). If ant was surrounded only by the pixels that belong to the background, at the next iteration of the algorithm, ant was randomly displaced to any of them. In every other case, ant was surrounded by the pixels that belong to the background, at the next iteration of the algorithm, ant was randomly displaced to any of them. In every other case, ant was surrounded by the pixels that belong to the background, at the next iteration of the algorithm, ant was randomly displaced to any of them.

According to the obtained value, and given parameters, ants decided if their current pixel belonged to the background or were at edge. Ant’s also had memory that prevented them to visit the same pixel again. Pheromone values were updated in a classical way:

\[ \tau_{i,j}(\text{new}) = (1 - \rho)\tau_{i,j}(\text{old}) + \sum_{k=1}^{m} \Delta \tau_{i,j}(k), \]  

\[(17)\]

where the amount of the pheromone left by the ant \(k\) on the edge pixel \((i,j)\) was \(\Delta \tau_{i,j}(k)\) and it was computed as:

\[ \Delta \tau_{i,j}(k) = \begin{cases} \eta_{i,j} & \text{if ant } k \text{ visited pixel } (i,j) \\ 0 & \text{otherwise} \end{cases} \]  

\[(18)\]

Termination criterion of the algorithm was maximal iteration number. After that edge image was obtained from the pheromone map by using thinning operation. The final result of the edge operation and its comparison to the Canny edge detector result is shown on the Figure. 11.

2) Particle Swarm Optimization: PSO is a population-based stochastic optimization technique modeled on the social behaviors observed in flocking birds. It was originally proposed by James Kennedy and Russell Eberhart in 1995 [47]. Since its inception, PSO has gained increasing popularity among researchers as a robust and efficient technique for solving difficult optimization problems. In PSO, individual particles of a swarm represent potential solutions, which move through the problem search space seeking for an optimal, or good enough, solution. The particles broadcast their current positions to neighboring particles. The position of each particle is adjusted according to its velocity and the difference between its current position, respectively, to the best position found by its neighbors and the best position it has found so far. As the model is iterated, the swarm focuses more and more on an area of the search space containing high-quality solutions.

The general form of PSO with inertia weight \((w)\) is given in Algorithm 3. Firstly, the PSO is made up of a large number of interacting elements (particles). Although the nature of the particles is simple, understanding the dynamics of the whole is nontrivial. Secondly, the particles are provided with memory and limited intelligence, which means that from one iteration to the next a particle may be attracted toward a new personal best position \(P_i\), or a new neighborhood best position \(P_G\), or both. Thirdly, the forces that effect particle’s behavior are stochastic (i.e. \(\phi_{1,t}\) and \(\phi_{2,t}\) are vectors whose elements are random numbers). Fourthly, the behavior of the PSO depends crucially on the structure of the fitness function. A little can be said about the useful function space in which to study the role of the fitness function in general. However, some progress has been made by considering simplifying assumptions such as isolated single individuals, search stagnation, or absence of randomness [48]. In the rest of this section we provide a little overview of PSO algorithms applied to face detection and verification problems. Those problems are not strictly optimization problems, but still, an optimization is what makes the results of face detection or verification problems better.

In the [49], [50] an algorithm for face detection based on the PSO generated templates has been described. For the illustration, we provide an extended overview of this PSO algorithm. The proposed algorithm uses a small set of segmented (normalized) frontal face images in order to set the initial PSO parameters (generate \(N\) particles, set their velocities and positions) and determine parameters \([x_{1}, x_{2}, y_{1}, y_{2}, \Delta y_{1}, \Delta y_{2}, T]\). The meaning of those parameters is depicted on Fig. 12. After that, directional image [51], [52] is computed for each face.
Algorithm 3 Classical PSO

1: Initialize a population array of particles with random positions and velocities on $N$ dimensions in the problem space: For each particle set its speed $v_i$ as a random number in the range $[-V_{\text{max}}, V_{\text{max}}]$ for each direction ($V_{\text{max}}$ is the maximum speed of the particles in $k_{th}$ direction), and position $x_i$ as a random point in the problem space.

2: loop

3: For each particle, evaluate the desired optimization fitness function in $N$ variables

4: Compare particle’s fitness evaluation ($f_i^t$) with its personal best fitness value $f_i^{\text{best}}$. If current value is better than $f_i^{\text{best}}$, then set $f_i^{\text{best}}$ equal to the current value, and $P_i$ equal to the current location $x_i^t$ in $N$-dimensional space.

5: Identify the particle in the neighbourhood with the best success so far (greatest value of $f_i^{\text{best}}$), and assign its position to the variable $\hat{P}_i$. That is, $P_i = P_{g(i)}$, where function $g(i)$ determines the index of the particle with the best so far fitness value that is located within the neighbourhood around the particle $i$.

6: Change the velocity and position of the particle according to the following equations

$$v_{i,t+1}^{(j)} = w v_{i,t}^{(j)} + \phi_{1,t} \circ (P_i - x_i^{(j)}) + \phi_{2,t} \circ (\hat{P}_i - x_i^{(j)})$$ (19)

$$x_{i,t+1}^{(j)} = x_i^{(j)} + v_{i,t+1}^{(j)}$$ (20)

7: If a criterion is met, exit loop

8: end loop

$\phi_{1,t}$ and $\phi_{2,t}$ are $N$-dimensional vectors whose elements are random numbers uniformly distributed in $[0, \phi_{\text{max}}]$. New random vectors are drawn for each particle $i$ and iteration $t$. The symbol $\circ$ represents a component-wise multiplication. Commonly, $\phi_{\text{max}} = 2$, but can change in different implementations.

Fig. 11: Edge detection by ACO algorithm [40] and Canny edge detector (all the images have size $512 \times 384$). The parameters for the ACO algorithm are: number of ants $N = 5050$, $\alpha = 2.5$, $\beta = 2$, $\rho = 0.001$, $q_0 = 0.95$, max. iteration number = 100. For the Canny algorithm parameters are: mask size $3 \times 3$, $\sigma = 1.4, 0.13$ and 0.38 lower and upper threshold. Canny edge detector produces better edge detection results.
Fig. 12: Simple model of the face used to determine parameters ([50])

\[ X_{i,l,m}(t + 1) = X_{i,l,m}(t) + v_{i,l,m}(t + 1) \]  \hspace{1cm} (23)

Evaluation function is given by:

\[ f_i^k = \frac{1}{N_R \cdot PM_i} \sum_{h=0}^{N_R-1} \sum_{l=0}^{N_w-1} \sum_{m=0}^{N_h-1} IM_{i,l,m} \]
\[ (I_{ref_{h,l,m}} \cdot \alpha_{i,h,l,m} - (1 - I_{ref_{h,l,m}})) \cdot \frac{\pi}{2} \]  \hspace{1cm} (24)

where \( I_{ref_{h,l,m}} \) indicates that the component of the face direction image \( h \) at position \( (l, m) \) is present (\( I_{ref_{h,l,m}} \) is equal to 1 or 0). \( N_R \) is the number of frontal faces, \( N_w \) and \( N_h \) are particle width and height, and \( PM_i \) is the total number of points of particle \( X_i \). Factor \( \alpha_{i,h,l,m} \) is called “angular similarity” and it is used to measure the angle difference at the specific coordinate in the PSO template and the angle at the directional image from face \( h \). It is defined as:

\[ \alpha_{i,h,l,m} = \frac{\pi}{2} - 2 \cdot \min(|MM_{i,l,m} - I_{h,l,m}|, \pi - |MM_{i,l,m} - I_{h,l,m}|) \]  \hspace{1cm} (25)

\( k \) is the iteration number.

Final PSO generated face templates are shown at Fig. 14.

Paper [53] analysed several schemes of visible and near infrared image fusion used for face verification. In each scheme, the PSO algorithm was employed to find the optimal fusion weights, that when employed on the test face database would yield as high as possible verification decisions (accept/reject scores). It is important to notice that images used in the experiments (the authors made database), were taken in the constrained conditions. Both images, visible and near infrared, were perfectly matched (the authors used a calibrated cameras setup that produce perfectly matched image pairs), so no geometric normalization was necessary. Another important factor is the database size. Authors stated that it contains images of 60 different persons, taken in two different sessions (time difference 5 weeks). Each session contains 30 image pairs per person with varied illumination conditions. Unfortunately, the authors haven’t made their database available, so the only image samples were those from the published papers. An example image pair is shown at Fig. 15. The fusion of such image pairs, should produce better verification results since visual image is very sensitive to illumination, and near infrared image can’t capture texture details as the visual.

To obtain fused image, 3 fusion schemes are proposed (schemes are depicted in Fig. 16): The first one (called image level fusion) is performed at the wavelet coefficients of each picture (after Discrete Wavelet Transform, (DWT) of each picture). The resulting fused image coefficients are calculated by:

\[ F_i = \alpha_i V_i + (1 - \alpha_i) N_i \]  \hspace{1cm} (26)

where \( V_i \) and \( N_i \) are \( i^{th} \) wavelet coefficients of visible and near infrared images. \( \alpha_i \in [0, 1] \) is \( i^{th} \) weight which value has
The left image is visual image, the image in the middle is near infrared image, and the right image is their fused image. The fused image have both good properties: it has textures copied from the visible image, and it is also more illumination-invariant due to the features of the near infrared image.

As an illustration of AOC system modeling and utilization for the image processing problems, we propose a new approach for detecting human faces from color images under

weights are determined by the PSO algorithm, such that $W_1 + W_2 = 1$.

The experimental results reported, that equal error rate (EER) value was significantly improved in each fusion scheme. In the first it was reported to be EER=1.75, second EER=2.42, and third EER=2.07, over the EER=8.47 when only visible face image was used (without fusion).

III. AOC IN IMAGE PROCESSING

As an illustration of AOC system modeling and utilization for the image processing problems, we propose a new approach for detecting human faces from color images under
complex conditions such as non-uniform illumination, arbitrary evolutionary image background, etc. The approach first utilizes AOC evolutionary computation technique to detect and locate the face-like regions. A number of color-sensitive entities are uniformly distributed in the two dimensional image environment to cluster the skin-like color pixels and segment each face-like region by activating their behaviors. After the face-like regions are located, the same AOC system is employed to find contours of all detected regions. According to the number of contours each region has, a feature called Euler number of each region is computed. The Euler number for an arbitrary image (previously segmented and labeled) is computed as:

$$EN = O - H,$$

(29)

where the $O$ is the number of objects, and $H$ is the number of holes. Here however, when dealing with the skin-like regions, we are analysing each skin-like region separately, hence the number of objects is always equal to 1. The number of holes, as well as, the Euler number of each skin-like region, is easily determined from the number of the contours of that region:

$$EN = 1 - (C - 1),$$

(30)

where $C$ is a number of contours of the analysed region. Each region with $H$ holes, have $C + 1$ contours (one around the whole object and one around each contour).

Experimental results show that the proposed approach is fast, robust and also has a high detection rate. The weakness we are analysing each skin-like region separately, the proposed approach is fast, robust and also has a high detection rate. The weakness

First we need to define entity properties and evaluation functions. For the purpose of skin-like color detection, entity state is defined in the same way as in introduction part of this paper: $S = (internal\_state, p, L_e)$, where the values for the $internal\_state$ can be active, passive, dead and communicable. $p$ is entity’s position and $L_e$ is entity’s lifespan (a maximal ages an entity can reach before dying). Notice that in this case entity doesn’t have memory since for this problem it is not necessary. The entity evaluation function should be color-sensitive. That is, when entity applies this function on a certain pixel location, this function will tell entity if the corresponding pixel has a color that is characterized as a human skin color. Prior to defining entity evaluation function, we need to find a cluster of colors in a color space that define human skin color.

Some research results [55] show that (1) human skin colors cluster in a small region in the RGB color space; (2) human skin colors differ more in brightness than in colors. Therefore, the normalized RGB model is considered to be capable of characterizing human faces with less variance in color.

Generally, colors of each pixel are expressed by the combination of $R$, $G$, $B$ components, and the brightness value $I = R + G + B$, where the range of each components value is $[0, 1, \ldots, 255]$. Since the color information is very sensitive to the brightness value of the pixel, each color component value can be normalized with the brightness value $I$ as follows:

$$r = R/I, \quad g = G/I, \quad b = B/I,$$

(31)

where $r + g + b = 1$.

In addition to the RGB model, it is commonly recognized that the HSV (hue, saturation, and value) model is more similar to the human perception of color [55]. The hue $H$ is a measure of the spectral composition of a color and represented as an angle, which varies from 0° to 360°. The saturation $S$ refers to the purity of colors, which varies from 0 to 1. The darkness of a color is defined by the value $V$, which ranges also from 0 to 1. The HSV color model can be converted from the RGB model using the following equations [56]:

$$H_1 = \arccos \frac{0.5(R - G) + (R - B)}{\sqrt{(R - G)^2 + (R - B)(G - B)}}$$

(32)

$$H = H_1 \quad \text{if } B \leq G$$

(33)

$$H = 360^\circ - H_1 \quad \text{if } B > G$$

(34)

$$S = \frac{\max(R, G, B) - \min(R, G, B)}{\max(R, G, B)}$$

(35)

$$V = \frac{\max(R, G, B)}{255}$$

(36)

After a large number of tests on the XM2VTS face image database, we have chosen the parameters as follows:

$$0.36 \leq r \leq 0.465, \quad 0 \leq H \leq 50,$$

$$0.20 \leq S \leq 0.68, \quad 0.35 \leq V \leq 1.0$$

According to these parameters entity evaluation function looks as:

$$F(p) = (0.36 \leq r(p) \leq 0.465) + (0 \leq H(p) \leq 50) + (0.20 \leq S(p) \leq 0.68) + (0.35 \leq V(p) \leq 1.0),$$

(37)

where $r(p)$ corresponds to the red component of the pixel at position $p$, that is $r(i,j)$. $H(p)$, $S(p)$ and $V(p)$ correspond to the Hue, Saturation and Value components and have the similar meanings. Formulation $(a \leq X \leq b)$ is evaluated as 1 if both inequalities hold for the value of $X$. Otherwise it is equal to 0. If the entity is located at the skin-like pixel its evaluation function will be equal to 4 (otherwise the result will be in $[0, 3]$).

The algorithm using an AOC approach to detect face-like regions in a color image can be described as follows:

1) Uniformly distribute an initial set of entities $E(0) = \{c_1\}$ over the image. $c_1$ represents one entity class, where $c_1 = (\{c_1\}, \mu_E, \mu_E(c_1) = N$, (set $E$ contains only one entity class with $N$ entities initially in it. All the entities are to detect skin-like pixels and have the same properties, hence one entity class is enough). In
order to detect all the possible faces, the entities are distributed one per each 20 × 20 part of the image, therefore, \( N \) is equal to the total number of pixels of the image/400. Store to all cells of the notice board their pixel position numbers. The top left pixel has number 0 and the bottom right pixel has number \( W \times H - 1 \) (see Fig. 20). Initial \( \text{internal\_state} \) of each entity is set to \textit{active} (that means \( \Phi(\text{active}) = N \)). Set time \( t = 0 \).

2) Execute algorithm 4. in order to determine skin-like regions

3) At this step all the entities are in the state \textit{passive}. Every such an entity has detected one skin-like pixel and kept positioned on it. Now we have to determine image regions. That is, every pixel that belongs to some region need to has the same label. For this purpose a pixel position number can be used (it was set in the step 1.). For the simplicity, we can decide that each region will have the smallest value of the pixel locations that it consists of as its label. For the region at figure 20 this value is equal to 7.

4) Change states of all entities to \textit{communicable}.

5) Execute algorithm 5. in order to determine labels of the detected skin-like regions.

6) Now all regions have the same label. Remove the small regions (e.g. regions whose number of pixels is smaller than 5\% of the number of pixels of the biggest region). Determine entities that are on the regions edge (for each region). Entities that are on the regions edge, have at least one pixel that is background (see Fig. 21). Set the \textit{internal\_state} of each entity on the edge to \textit{communicable}. Now, we need to detect number of the contours for each region. To do this, similarly to the region label detection, we will determine label of the contour for each contour of the analysed region (we know that of all pixels that are on a given contour, one will have the smallest pixel location number). It is important to emphasize that we are using different pixel location numbers for to determine region labels and contours. Initially, all the numbers (for regions and contours) are set to their raster position values.

7) For each detected region execute algorithm 6. in order to determine labels of each contour of the analysed region. According to the number of those labels, region Euler number is computed. Note that algorithm 6 is almost equal to the algorithm 5. The only difference is that in former only the entities on the region edge are considered.

Examples of the proposed AOC algorithm are shown at the figures 18. and 19.

We have experimentally determined that each skin-like region which Euler number is in \([-5, -20]\) can be classified as a potential face region. That means that lots of non-face regions will be eliminated that way, but some non-face regions can still be treated as a face-like regions. To minimize detection error, we need to locate other face features, such as eyes, nose or lips. However, detection of those facial parts is beyond this overview.

IV. AOC HARDWARE AND SOFTWARE ENVIRONMENTS

Depending on the available hardware and software environments, we can consider several approaches of an AOC-based system implementation and evaluation. First, and the most obvious approach is to build program simulator that runs on a single processor machine and supports virtual parallelism needed to simulate entity activities. Such an approach is suitable for analysing system functionality and assessing only the approximate performance values since all concepts run virtually and therefore are simplified.

- Program simulator
- Using sistolic arrays (of processors)
- Using massively parallel (graphic) processors
- Building custom FPGA based architecture Such hardware implementations are the fastest ones, but also very

### Algorithm 4 AOC skin-like region detection

1: while \( \Phi(i)(\text{active}) > 0 \) do
2:   for all entity \( e \) in \( \Phi(i) \) do
3:     if \( e \) has \( \text{internal\_state} = \text{dead} \) then
4:       remove this entity from the environment \( \Phi(i) \)
5:     end if
6:     if \( e \) has \( \text{internal\_state} = \text{passive} \) then
7:       skip this entity. Chose another entity from \( \Phi(i) \)
8:     end if
9:     if \( e \) has \( \text{internal\_state} = \text{active} \) then
10:    execute evaluation function \( \Phi(i) \)
11: end if
12: if \( \Phi(i) = 4 \) (i.e. entity is positioned on the skin-like pixel) then
13:   change \( \text{internal\_state} \) of entity \( e \) to the \textit{passive}.
14: end if
15: if \( \Phi(i) < 4 \) (current pixel is not a skin-like pixel) then
16:   move to another adjacent (within 8-neighbourhood) free pixel location. If non of the pixel locations is free then wait this time step. Increase ages of the entity \( e \) by 1.
17: end if
18: if ages of entity \( e \) exceeds its lifespan \((\text{ages} \geq L_s)\) then
19:   change entity’s \( \text{internal\_state} \) to \text{dead}.
20: end if
21: Increase time step \( t \leftarrow t + 1 \)
22: end for
23: end while
This page contains a table and images related to face detection and region labeling. The text describes the process of identifying and labeling regions in an image, with a focus on the use of the AOC algorithm. The table shows pixel locations and corresponding pixel numbers, and the images illustrate the detection of skin-like regions and contour detection.

**Algorithm 5** Determine skin-like regions’ labels

```plaintext
1: while \( \Phi^{(t)}(\text{communicable}) > 0 \) do
2:     for all entity \( e \) in \( \mathcal{E}^{(t)} \) do
3:         if \( e \) has \( \text{internal\_state} = \text{passive} \) then
4:             skip this entity. Choose another entity
5:         end if
6:         if \( e \) has \( \text{internal\_state} = \text{communicable} \) then
7:             check 8-neighbourhood for the pixel location values. If any of those values is greater than the entity’s current value, change their value to the entity’s current value. Do this change only if another entity is located on such pixel. Also set the internal state of those entities (whose location number has been changed) to \text{communicable}. Set the \text{internal\_state} of the entity \( e \) to \text{passive}.
8:         end if
9:     end for
10: end while
```

Specialized and suitable for only one or a few processing functions. Depending on the algorithm, processing and memory capabilities needed to be performed by each entity could vary, and some of them are too complex for FPGA implementations on each entity (e.g., computing square roots, trigonometric functions, derivatives).
Algorithm 6 Determine contour labels of each skin-like regions

1: while $\Phi^{(t)}(\text{communicable}) > 0$ do
2:   for all entity $e$ in $\mathcal{E}^{(t)}$ do
3:     if $e$ has internal state = passive then
4:       skip this entity. Choose another entity. $E$
5:     end if
6:   if $e$ has internal state = communicable then
7:     check 8-neighbourhood for the pixel location values. If any of those values is greater than
8:     the entity's current value, change their value to the entities current value. Do this change
9:     only if another entity is located on such pixel and if that other entity is also on the edge. Also
10:    set the internal state of those entities (whose location number has been changed) to communicable.
11:   end if
12:   Increase time step $t \leftarrow t + 1$
13: end for
14: end while

V. Conclusion

Most image processing and feature extraction algorithms perform search with a given mask by sliding search window from one image edge to another column by column and row by row, until the whole image is not evaluated. If desired optimality criterion is not achieved (i.e. feature is not detected) at any search window location, they inherently scale search window to greater sizes, and perform the whole search process again. The process terminates when desired features are detected or when the search window size exceeds some predefined limit.

In AOC based approach, image is searched in a different fashion. A number of autonomous agents entities distributed over the image environment, perform search for a given feature or pattern concurrently at many pixel positions. Entities' search space, referred usually as the local search space is square shaped window that can vary in size. The downside of this approach is that such search strategy doesn’t ensure that the whole image will be evaluated. Another downside is that some parts of the image can be evaluated more than once. To overcome those problems, we can distribute entities uniformly over the image in a such density that we can guarantee that each part of the image will be evaluated and feature detected (i.e. if the image feature is small we will use more entities, otherwise if the image feature occupies lots of pixels on the image, smaller number of entities will suffice). All this is only important for the initialization phase. The optimal number of entities will be adjusted automatically though the entity’s aging ability.

This paper presented an overview of the AOC-based systems properties, and showed how such systems can be utilized for the image processing problems. Algorithms based on the ant colonies or swarms of the particles, that have very similar structure as the AOC approach, also presented that, because of
their optimization characteristics, are very suitable for optimal feature extraction. They are suitable for every processing step where an optimality criterion of feature selection has to be met. Comparison between entities of the AOC, ACO, PSO and CA systems are provided in the table II.

As we consider, the benefits of an AOC system can be summarized as the following:

- it is distributed (and almost completely parallel) by the nature
- biologically inspired (there are numerous life forms and spcies whose known behaviors and characteristics can be computationally modeled and used as an inspiration for solving computational problems)
- the same computational model can be used to solve different problems (image processing, optimization)
- it doesn’t have to search the whole problem space in order to find the optimal or good enough solution (the risk is that we will find suboptimal solution instead. This (heurist property) can be considered as a downside of all evolutionary algorithms).
- due to dualism between the software and hardware engineering, we can expect that the concepts and algorithms developed as a part of the AOC application on image processing problems could be easily transmitted to their hardware counterparts (one such try is described here [17], however not using the AOC notation)

For the downsides of the AOC approaches, we consider these:

- they are not computationally efficient if utilized on sequential computational architectures (some processing time is wasted in modelling and simulating AOC processing model)
- they need different computer architecture. Present algorithms are mostly based for sequential execution as the sequential computer architectures are the most common. However, this trend has changed during the last 10 years as parallel architectures (e.g. using graphic card processors - GPU using CUDA programming environment) are available and actively explored. Such architectures are more suitable for the AOC development.
- they request difficult problems (easy tasks already have simple and efficient solutions). There is no need for parallelization of the short or simple problems, since there is a great possibility that the final solution will be worser to the sequential one due to synchronization/communication overhead.

REFERENCES

<table>
<thead>
<tr>
<th>AOC entity (agent)</th>
<th>ACO ant</th>
<th>PSO particle</th>
<th>CA automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>characterized by</td>
<td>position, age, state, evaluation function</td>
<td>position, memory (about crossed path)</td>
<td>position, velocity, fitness function</td>
</tr>
<tr>
<td>appears in heterogeneous or homogeneous structures</td>
<td>heterogeneous and homogeneous</td>
<td>usually homogeneous</td>
<td>homogeneous</td>
</tr>
<tr>
<td>modify its evaluation function it time</td>
<td>yes (have ability to adapt and learn)</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>have memory</td>
<td>yes</td>
<td>yes</td>
<td>usually no (application dependent)</td>
</tr>
<tr>
<td>use local or global problem data (have information about the problem in the whole or just part of it)</td>
<td>local data (emphasized)</td>
<td>local data</td>
<td>global and local data</td>
</tr>
<tr>
<td>can move (change position)</td>
<td>yes/yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>can cease to exist / reproduce (use aging)</td>
<td>yes/yes</td>
<td>usually no/no</td>
<td>no</td>
</tr>
<tr>
<td>can communicate with other entities</td>
<td>yes directly and indirectly (through the notice board)</td>
<td>indirectly (by pheromone trail), possibly directly</td>
<td>only indirectly</td>
</tr>
<tr>
<td>autonomous</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>