On the optimization of systems using evolutionary algorithms and techniques

Itshak Tkach and Tim Blackwell

Goldsmiths, University of London, UK

In this chapter, evolutionary computation techniques, algorithms and research are presented for the optimization and allocation problems. Several aspects of continuous optimization, systems security and supply networks (SN) are illustrated. The real-life optimization and security problems in systems, automation, SN and law enforcement are NP-hard optimization problems, thus evolutionary algorithms (EA) that employ meta-heuristic methods are useful for solving them. EA gain significant interest in recent years, and this chapter summarizes some of the advances in that field and then summarizes their applications for real-life problems. The rest of this chapter is organized as follows.

First, the introduction of the developments of nature-inspired EAs is described. Then the working principles of genetic algorithms (GA), swarm intelligence, and other nature-inspired optimization algorithms are given. Next, the overview of the various applications that were solved and optimized by EAs is presented.

The reader of this chapter will be familiar with the following topics:

The state-of-the-art EA in AI and their working principle.

The way to harness EA for optimization and finding optimal solutions.

Function and multi-dimensional optimization.

Controlling and optimizing a collaborative system in real-time while addressing several tasks in a complex environment.

Introduction

Evolutionary algorithms are nature-inspired techniques that are a subfield of computational and artificial intelligence (AI). EA uses an evolutionary process within a computer to address complex engineering problems that require an exhaustive search that traditional algorithms are unable to solve in a finite amount of time. Part of EA is swarm (SI) intelligence algorithms, that employ the emergent collective intelligence of simple agents to solve complex problems.

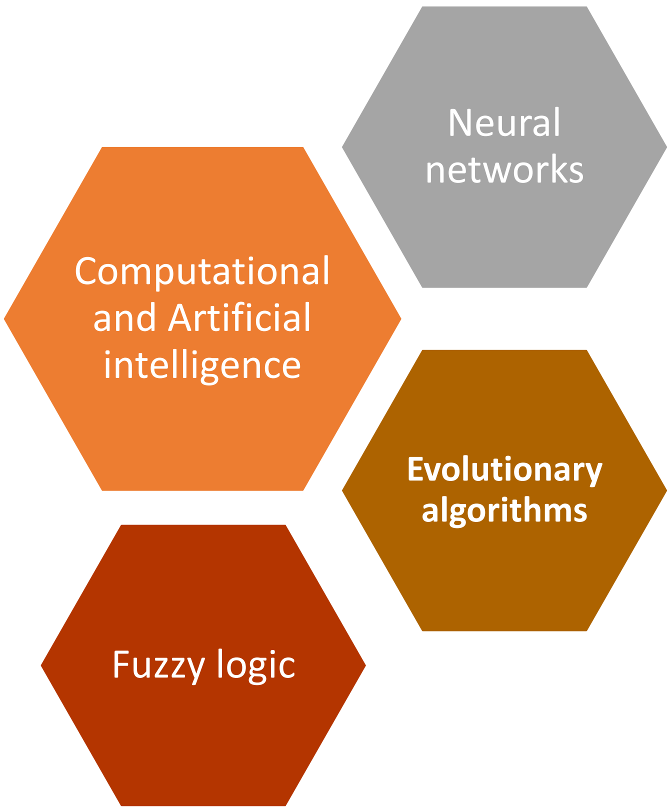

Figure 1: Computational intelligence disciplines; Neural networks, Evolutionary algorithms and fuzzy logic

EA has been developed since the early 1960s. Evolution strategies (ES) were introduced (Rechenberg, 1973) and (Schwefel, 1974). Evolutionary programming (EP) was first used by (Fogel, 1963) to use simulated evolution as a learning process using finite-state machines. Genetic algorithms (GA) became popular through the work of (Holland, 1975) with studies of cellular automata and formulation of the next generation. Genetic programming (GP) is an extension of GA for program evolution which was introduced by (Koza, 1990). Simulated annealing (SA) was developed by (Kirkpatrick, et al., 1983). It is a probabilistic algorithm inspired by the annealing of metals in metallurgy. Ant colony optimization (ACO) proposed by (Dorigo, 1992) is a probabilistic optimization technique based on the foraging behaviour of ants seeking the shortest path between their colony and a source of food. (Kennedy & Eberhart, 1995) introduced particle swarm optimization (PSO) which is a computational method for optimising problems by using a population of particles moving around the search space according to the simple principle of particle's position and velocity. The bees algorithm (BA) was formulated by (Pham, et al., 2005) based on the foraging behaviour of honey bees. The algorithm exploits global explorative search with local exploitative search. The artificial bee colony (ABC) algorithm is a metaheuristic introduced by (Karaboga, 2010), as an extension of BA, it has three types of bees, employed bee, onlooker bee and scout bee. Heterogeneous Distributed Bees Algorithm (HDBA) is a multi-agent metaheuristic algorithm initially introduced by (Tkach, et al., 2013). It enables solving of combinatorial optimization problems with multiple heterogeneous agents that possess different capabilities and performances. The main evolutionary algorithms are summarized in Figure 2.

Based on the Google Scholar Metrics (2022), the top 6 publications are IEEE Congress on Evolutionary Computation, Swarm and Evolutionary Computation, Conference on Genetic and Evolutionary Computation, Evolutionary Computation, International Conference on Applications of Evolutionary Computation, International Conference on Genetic and Evolutionary Computing (ICGEC).

1960	Random Search	
	ES	EP
1970		
	GA	
1980		
	SA	Tabu Search
1990	GP	ACO
	PSO	
2000		
	BA	ABC
2010		HDBA

Figure 2: Timeline of ET

The main EA and their working principles

Genetic Algorithm

A genetic algorithm (GA) is an evolutionary algorithm that is used for solving optimization problems with non-polynomial complexity. It was introduced by John Holland in 1975 (Holland, 1975), as a search optimization algorithm based on the mechanics of the natural selection process. The basic concept of this algorithm is to mimic the concept of the survival of the fittest. The evolution usually starts from a population of randomly generated individuals consisting of legitimate candidate solutions, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified by crossover and mutation to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory level of the objective function has been reached.

Pseudocode of GA

```
 Initialize population
       Generate random genotypes representing legitimate solutions
       While not terminated do
              For each chromosome from the population
                     Compute fitness functions $V_I$
              Make next population
                     Select parents
                     Apply crossover by recombining pairs of parents
                     Apply mutation to offspring by changing an allocation in a genotype
              Evaluate new solutions by computing fitness functions $V_I$
              Store the chromosome that obtains the best fitness function
       End while
  Take the stored chromosome
```

Ant colony optimization

An ant colony swarm algorithm called Ant Colony Optimization (ACO) was developed by (Dorigo, 1992). It is inspired by the behaviour of ants foraging in nature. Research on the behaviour of ants shows that most of the information transmission between individuals in a group and between individuals and the environment is carried out by the chemical substances produced by ants. This is a special substance called pheromone. They use a pheromone to mark the path on the ground, such as the path from a food source to an ant colony. When ants walk from a food source to an ant nest or from an ant nest to a food source, they will release pheromone on the ground they pass by, thus forming a path containing pheromone. Ants can perceive the concentration of pheromone on the path, and select the path with the highest pheromone concentration with a higher probability. Ants find the location of food along the way by sensing the pheromone released by other ants. This way of influencing the path selection of ant colonies based on the information of chemical substances released by other ants is the inspiration for ACO. It has been widely recognized and its application has been extended to all aspects of the optimization problem field. In ACO, each ant is an agent that chooses a solution with a probability that is a function of the performance and the amount of trail laid. To force the ant to make legal selections, transitions to already visited sensors are disallowed until a cycle is completed (this is controlled by a *tabu list*); when it completes a cycle, it lays a substance called a trail on each path visited. After several generations, the algorithm converges to the best path, which hopefully represents the optimum or suboptimal solution to the problem.

$$\upsilon_i(t+n) = \rho \times \upsilon_i(t) + \Delta\upsilon_i \geq \upsilon_0$$

where υ_i is the intensity of ant's *i* trail at time *t*. *n* is the number of sensors (every *n* iterations – cycle, each ant completed a cycle). ρ is a coefficient such that $(1 - \rho)$ represents the evaporation of trail between time *t* and *t*+*n*, υ_o is the threshold.

$$\Delta\upsilon_i = \sum_{k=1}^{m} \Delta\upsilon_i^k$$

where $\Delta\upsilon_i$ is the quantity per unit of trail laid on sensor *i* by the *k*-th ant between time *t* and *t*+*n*; it is given by:

$$\Delta\upsilon_i^k = \begin{cases} \dfrac{Q}{L_k} & \text{if k-th ant uses path i in its cycle} \\ 0 & \text{otherwise} \end{cases}$$

where *Q* is a constant and L_k is the tour length of the *k*-th ant. *Tabu list* saves the sensors already visited up to time *t* and forbids the ant from visiting them again. When a tour is completed, the *tabu list* is used to compute the ant's current solution (i.e., the set of sensors being reached).

The probability of going to sensor *i* for the *k*-th ant is given by:

$$p_i^k(t) = \begin{cases} \dfrac{\upsilon_i^{\psi}(t) \times \xi_i^{\omega}}{\sum\limits_{k \in allowed_k} \upsilon_i^{\psi}(t) \times \xi_i^{\omega}} & \text{if i} \in \text{allowed}_\text{k} \\ 0 & \text{otherwise} \end{cases}$$

where, visibility ξ_i is quantity *S*, *allowed*$_k$ is the set of sensors not in *tabu list*, ψ and ω are parameters that control the relative importance of trail versus visibility.

The stop criterion is given by a threshold that quantifies the desired performance of the sensors for a given task. The optimal sensor allocation is given by:

$$V_{OS} = \sum_{k \in tabu \text{ list}} S_j \geq threshold$$

where *j* is the index of sensors in the *tabu list.*

Pseudocode of the ACO algorithm

```
1. Initialize:
       Set $t$=0
       $\Delta v_i^k$=0
       Place $N$ ants on sensors
2. If end of mission, then go to step 5
       else
           Clear tabu list
           Upon task arrival calculate the new task priority $f_i$
           Update queue of tasks based on their priorities
           Go to step 3
3. While $\Delta_i$>0:
       If new task arrived, then go to step 2
               For $k$=1 to $N$ do
               Choose the sensor $i$ with probability $p_i^k(t)$
               Insert sensor $i$ to tabu list
4. If ($t_{ex_i}$=0) then set $\Delta_i$=0 and go to step 2
       else
               Go to step 4
5. Finalize:
       Calculate $V_I$
       Calculate $T$
       Stop
```

Particle swarm optimization

Tim – LPSO and GPSO

Particle swarm optimization (PSO) is a swarm intelligence optimization algorithm first proposed by (Kennedy & Eberhart, 1995). The algorithm uses swarm iterations, allowing particles to follow the optimal particle search in the solution space to simulate the mutual cooperation mechanism of the foraging behaviour of groups of birds and fish to find the optimal solution to the problem. All particles are searched in *D*-dimensional space. All particles are determined by a fitness function to determine the fitness value to judge the current position. Each particle must be endowed with a memory function to remember the best position found. Each particle also has a speed used to determine the distance and direction of flight. This speed is dynamically adjusted based on its own experience and peer flight flying experience. The *d*-dimensional velocity update formula of particle *i*:

$$\boldsymbol{v}^k_{id} = w\, \boldsymbol{v}^{k\text{-}1}_{id} + c_1 r_1 (pbest_{id} - \boldsymbol{x}^{k\text{-}1}_{id}) + c_2 r_2 (gbest_d - \boldsymbol{x}^{k\text{-}1}_{id}) \tag{21.2}$$

$$\boldsymbol{x}^k_{id} = \boldsymbol{x}^{k\text{-}1}_{id} + \boldsymbol{v}^k_{id} \tag{21.3}$$

where $\boldsymbol{x}^k_{id}$ is the *d*-dimensional component of the flying velocity vector of particle *i* in the *k*-th iteration. $\boldsymbol{v}^k_{id}$ is the *d*-dimensional component of the position vector of the particle *i* in the *k*-th iteration. c_1, c_2 are acceleration constant, adjust the maximum step length of learning. R_1, R_2 are two random function, in the range [0,1], in order to increase the search randomness. *w* is a inertia weight, non-negative, adjusting the search range of the solution space.

(Shi & Eberhart, 1998)introduced the inertia weight ω and proposed to dynamically adjust the inertia weight to balance the global convergence and convergence speed. This algorithm is called the standard PSO algorithm. The inertia weight ω describes the influence of the particle's previous generation velocity on the current generation velocity. The speed update formula is as follows:

$$\boldsymbol{v}_{id} = K[\boldsymbol{v}_{id} + \varphi_1 r_1 (pbest_{id} - \boldsymbol{x}_{id}) + \varphi_2 r_2 (gbest_d - \boldsymbol{x}_{id})] \tag{21.4}$$

where the shrinkage factor K is ω limited by $\varphi_1$and φ_2. φ_1 and φ_2 are models that need to be set in advance parameters. The shrinkage factor method controls the behaviour of the system and finally converges, and can effectively search for different areas.

Pseudocode PSO

Differential evolution

Differential evolution (DE) is a heuristic algorithm based on population. It shares some similarities to GA since it employs similar operators; crossover, mutation, and selection, where DE relies on mutation operation while GA relies on crossover operation. This algorithm was introduced by (Storn & Price, 1997). Since this algorithm relies on mutation operation, it utilizes the mutation as a search mechanism and takes advantage of the selection operation to direct the search towards the potential regions in the search space. It aims to find the optimal solutions to possibly nonlinear and non-differentiable functions in continuous space.

DE starts from a set of randomly initial solutions, which is called population P. Each individual in P represents a solution, we define it as x. And the number of individuals is N. After a certain number of generations of evolution, DE could find the global optimal solution. DE includes crossover, mutation, and selection operators. Different from other optimization algorithms, the evolution of the DE algorithm is reflected by the differential information of multiple individuals. The mutation of DE is to randomly select two different individuals in the population and then scale their vector difference to perform vector synthesis with the individual to be mutated. The formula is as follows:

$$\boldsymbol{v}_r(t+1)=\boldsymbol{x}_{r1}(t) + F\,(\boldsymbol{x}_{r2}(t) - \boldsymbol{x}_{r3}(t)) \tag{21.5}$$

In which r_1, r_2, r_3 belongs to $[1, N]$, and $r_1 \neq r_2 \neq r_3$. F is the scale factor, t represents a generation. $\boldsymbol{x}_{r2}(t)$ - $\boldsymbol{x}_{r3}(t)$ is the difference, which tends to adapt to the natural scales of the objective landscape through the iteration of population. For example, the difference will become smaller when a variable of the population becomes compact. And this kind of adaptive adjustment helps speed up the exploration of solution space, which makes DE more effective. The operation of crossover operation is as follows:

$$u_\gamma(t+1) = \begin{cases} v_\gamma(t+1), \text{if } U(0,1) \geq CR \\ x_\gamma(t), \text{ otherwise} \end{cases} \tag{21.6}$$

where $U(0,1)$ represents a random real number uniformly distributed between [0,1], and CR represents crossover probability.

In DE, the strategy of greedy selection is adopted, which means when an offspring is generated, its fitness value of it is compared with the corresponding parent, and the individual with the better fitness will be selected to enter the next generation. The selection formula is as follows:

$$x_\gamma(t+1) = \begin{cases} u_a(t+1), \text{if } f(u_\gamma(t+1)) \le f(x_\gamma(t)) \\ x_\gamma(t), \text{ otherwise} \end{cases} \tag{21.7}$$

where *f* represents fitness function, and the goal is to minimize the value of fitness.

DE uses three control parameters: the population size *N*, the crossover possibility *CR*, and the scale factor *F*.

Pseudocode of DE

```
• Choose the parameters $CR \in [0,1]$ , and $F \in [0,2]$.
• Initialize all agents x with random positions in the search space.
• Until a termination criterion is met (e.g. a number of iterations performed, or
  adequate fitness reached), repeat the following:
    o For each agent x in the population do:
        ▪ Pick three agents a, b and c from the population at random,
          they must be distinct from each other as well as from
          agent x. (a is called the "base" vector.)
        ▪ Pick a random index $R \in \{1,\ldots,n\}$ where n is the
          dimensionality of the problem being optimized.
        ▪ Compute the agent's potentially new
          position $y=[y_1,\ldots,y_n]$ as follows:
            ▪ For each $i \in \{1,\ldots,n\}$ , pick a uniformly

              distributed random number
            ▪ If $r_i < CR$ or $i=R$ then set $y_i=a_i+F(b_i-c_i)$ otherwise set $y_i=x_i$. (Index position R is
              replaced for certain.)
        ▪ If $f(y) \le f(x)$ then replace the agent x in the population with
          the improved or equal candidate solution y.
• Pick the agent from the population that has the best fitness and return it as the best
  found candidate solution.
```

Simulated Annealing

Simulated Annealing (SA) algorithm is a heuristic algorithm, which simulates the process of annealing in metallurgy introduced by (Kirkpatrick, et al., 1983). The method models the physical process of heating a material and then controlled cooling to alter its physical properties, thus minimizing the system energy. Likewise, SA accepts a worse solution than the current solution with a certain probability, so it has more possibility to jump out of the local optimal solution and find the global optimal solution.

Pseudocode of SA

```
  Create initial solution Sol
```

```
Initialize temperature $t$
repeat
  for $i = 1$ to iteration-length do
    Generate a random transition from $Sol$ to $Sol_i$
    if $Cost(Sol) \geq Cost(Sol_i)$ then
      $Sol = Sol_i$
    else if $e^{\frac{Cost(Sol)-Cost(Soli)}{current\ temperature}} >$ random [0,1) then
      $Sol = Sol_i$
    end if
  end for
  Reduce temperature $t$
until no change in $Cost(Sol)$
return $Sol$
```

Heterogeneous Distributed Bees Algorithm

HDBA is a swarm intelligence algorithm that enables to solve combinatorial optimization problems that include multiple heterogeneous agents that possess different capabilities and performances. It was developed by (Tkach, et al., 2018). HDBA uses a probabilistic technique taking inspiration from the foraging behaviour of bees.

In this algorithm, each agent is represented as a ‘bee’, and agent utility, p_{ik}, is defined as a probability that the agent k is allocated to the task i and depends on both priority and the distance of the task/target from the sensor:

$$p_{ik} = \frac{F_i^{\alpha}\left(\frac{1}{D_{ik}}\right)^{\beta}}{\sum_{j=1}^{M} F_j^{\alpha}\left(\frac{1}{D_{jk}}\right)^{\beta}} \text{ if } \Delta_i > 0$$

where α and β are control parameters that bias importance of the priority and distance respectively $(\alpha, \beta > 0; \alpha, \beta \in \mathbb{R})$. The probabilities p_{ik} are normalized, and it is easy to show that:

$$\sum_{i=1}^{M} p_{ik} = 1$$

The HDBA decision-making mechanism uses a wheel-selection rule, where each agent has a probability with which it is allocated to the task from a set of available tasks. Once all the agents' utilities are calculated, each of them selects a task by “spinning the wheel".

The HDBA function of agents’ utility is developed for the case of heterogeneous agents with different performances. This setting is assumed to improve system performance as it can correlate the agents’ utility function with the value of their performances.

In order to define HDBA, a task's utility value (V_{ik}) as a function of the agent’s performance on that task is defined. When an agent receives information about the available tasks it calculates its performance for that task. The agent’s utility function is updated accordingly, and depends on the priority of the task, the distance from the task and the agent’s performance on that task:

$$p_{ik} = \frac{F_i^{\alpha}\left(\frac{1}{D_{ik}}\right)^{\beta} V_{ik}^{\gamma}}{\sum_{j=1}^{M} F_j^{\alpha}\left(\frac{1}{D_{jk}}\right)^{\beta} V_{jk}^{\gamma}} \text{ if } \Delta_i > 0$$

where γ is a control parameter that biases the importance of the agent's performance and V_{ik} is the performance of sensor k on task i.

<u>Pseudocode of HDBA</u>

```
1. Initialize:
      Set $t$=0
      Place $N$ bees on agents
2. If termination condition met then go to step 5
      else
         Upon task arrival calculate the new task priority $F_i$
         Calculate distances of agents from task $D_{ik}$
         Calculate performances of agents on tasks $V_{ik}$
         Go to step 3
3. If new task arrived then go to step 2
      For $i$=1 to $M$ do
            For $k$=1 to $N$ do
            Calculate probabilities for each agent $p_{ik}$
      Apply wheel-selection rule
   Allocate agents according to the selection
4. If ($t_{ex_i}$=0) then set $\Delta_i$=0 and go to step 2
      else
           Go to step 3
5. Finalize:
      Calculate $V_I$
      Calculate $T$
      Stop
```

Summary of implementation to optimization problems

System's resource and task allocation

Systems typically consist of multiple resources and tasks that need to be handled with different performances. An example of a sensory system was described in (Tkach & Edan, 2020). In this system the performance of the sensor is defined a priori based on the sensor's features, namely detection distance, resolution, and response time. Each sensor can only be allocated to one task at any given time and can be reallocated to another task at any moment. The priority of the task is an application-specific scalar value, where a higher priority value represents a task that has higher importance and must be attended to faster than other tasks. Higher priority tasks also have a higher benefit for completing them. Examples of such tasks include surveillance (gathering information on desired objects), security monitoring (preventing theft of goods and threats, and fire monitoring (forest fire detection and protection), among many others.
This system must deal with the real-time detection of unpredictable, unknown tasks arriving at unknown times and locations. The task occurrence is dynamic and unpredictable with different levels of importance of each task and must be detected as fast as possible. The sensors must be allocated to the tasks as fast as possible. The goal is to allocate each sensor an appropriate task at an appropriate time (Figure 3) and to ensure all tasks are completed in minimum time.

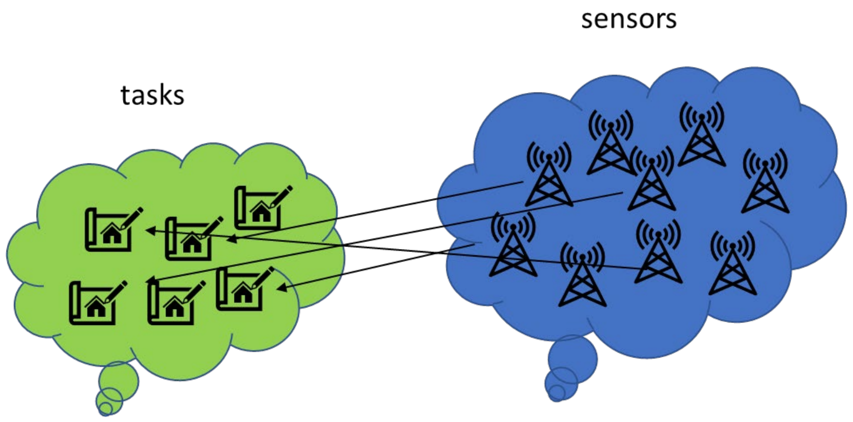

Figure 3: Distributed multi-sensor system allocation scheme.

When there are several tasks that require the same sensors, the allocation depends on the sensors' availability and performance, the physical distance of sensors from the tasks, and the priorities of individual tasks.

Optimization of geometric problems

Tim

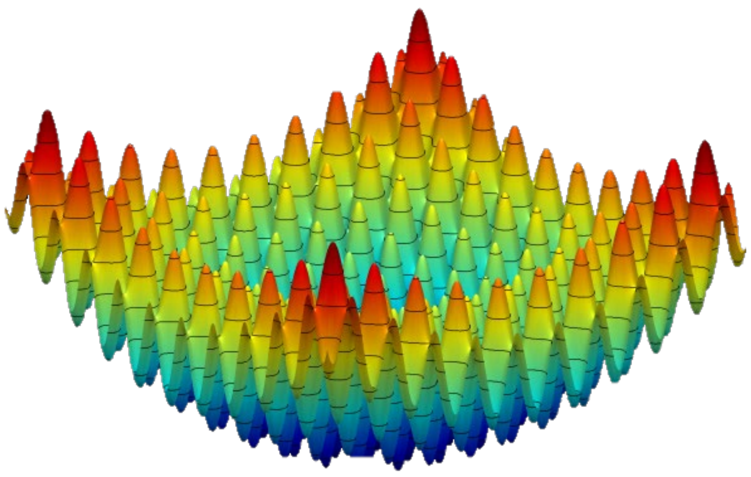

Figure 4: An example of continuous optimization function – 3-dimentioanl Rastrigin function.

Optimization of supply network security using sensory systems

Task administration protocols (TAPs) were used to overcome uncertainties and disturbances (i.e., failures, conflicts in priorities) in supply networks (Tkach, et al., 2017). TAPs consist of four protocols, one of which applies a bio inspired HDBA for sensors allocation and real time detection of targets for security. The role of this algorithm is to efficiently allocate high number of sensors to upcoming targets to detect as much targets as fast as possible.
Optimal sensors availability related to their monetary cost in the system was achieved by deployment of redundant sensors. Employing TAPs which use HDBA allowed dynamic, real-time allocation of distributed sensors to targets when they occur.

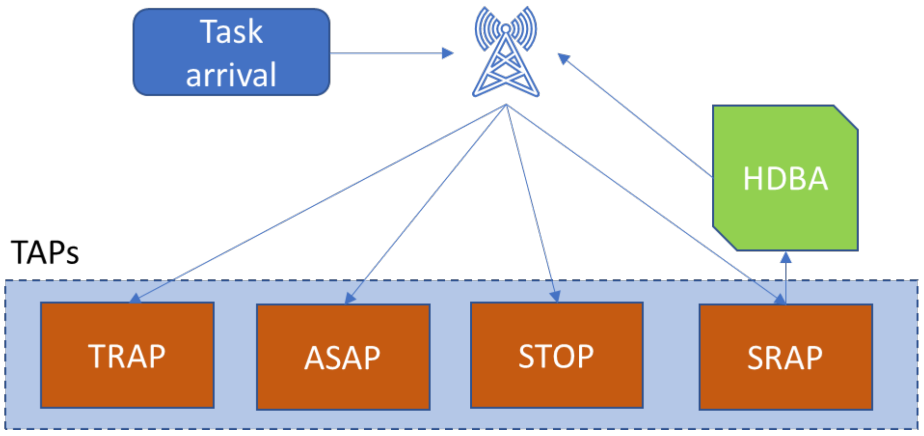

Figure 5: System procedure scheme

The system was designed as a dual-layer network; a process layer and a monitoring layer. The process layer consists of multiple sensors and is responsible for allocating them to complete tasks. As multi-sensor systems are vulnerable to some risks and problems, the monitoring layer functions at a higher level than the process layer and is used to monitor those problems in the process layer by applying TAPs to handle them. The first problem has to do with tasks that require very long attendance times. These tasks may occupy the sensors allocated to them for long time durations, thus delaying the handling of other tasks by those sensors. In this overloading problem, the sensors are overloaded with a portion of tasks which delays them. A time out policy, which recognizes if a sensor is experiencing a delay while other tasks are waiting for execution, can overcome this problem. The second problem that the sensory system must overcome relates to tasks that may have much higher priority over other tasks or the same priority as other tasks. A similar problem may arise when, due to malfunction or recognition problems, tasks are perceived by the system to be of a higher priority than they should be. Such tasks may unnecessarily occupy some sensors. The system may need to reprioritize those sensors which are unable to perform other tasks that must be handled quickly, ahead of less urgent tasks, due to being close to their deadline. In this deception problem, sensors are occupied and delayed by some tasks which may be neither urgent nor important. The third problem that the sensory system must overcome is related to a failure of a portion of sensors in the system. In this tampering problem, sensors may fail due to internal properties (e.g. hardware reliability) and external reactions (e.g. weather conditions). The system should be able to monitor these problems to ensure that system performance is unharmed.

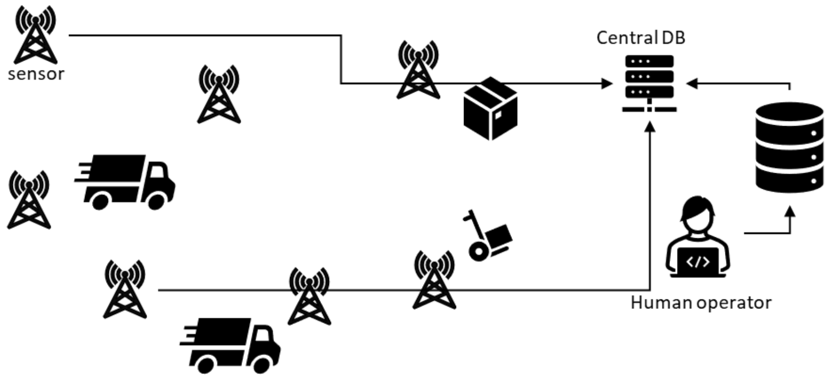

Figure 6: An illustration of a dual-layer logistic system with distributed sensors to monitor security.

This example presented a dual-layer system for applying task allocation algorithm and task administration protocols for efficient target detection for SN security that can deal with problems in the allocation process and a protocol for analysing the status of sensors to modify the allocation if necessary. It repeatedly identifies the current state of the system and takes proper actions to deal with allocation problems and improve system performance.

Optimization of dynamic multi-agent task allocation in law enforcement

EAs were implemented by (Tkach & Amador, 2021) for solving a realistic Law enforcement problem by employing HDBA to FMC_TAH+ and SA. This was a multi-agent problem with heterogeneous skills that work together on tasks to share the workload and improve response time. The workload associated with each task, indicating the amount of work to be completed for the incident to be processed was different. EAs allocated agents to dynamic tasks whose locations, arrival times, and importance levels are unknown a priori. The employed methods were compared to different performance measures that are commonly used by law enforcement authorities. This evaluation was shown to be effective in allocating dynamic tasks to heterogeneous police agents.

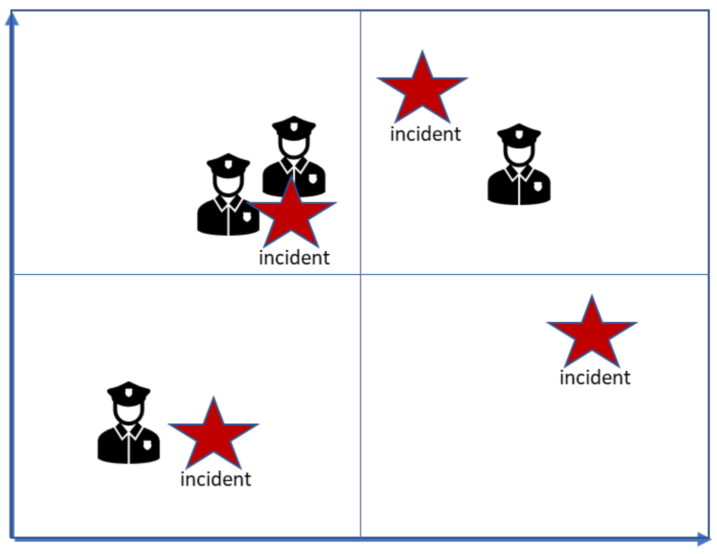

Figure 7: An example of police officers to incidents/task allocation problem.

Optimizer benchmark

A novel benchmark framework with known geometric properties and critical point topologies was introduced in by (Tkach & Blackweel, 2022). This benchmark tackled the problem of existing methods to analyse algorithm performance due to their inherent complexity. The benchmark made a realization of a specified barrier tree function in which funnel and basin of attraction geometries, and values and locations of all critical points are predetermined. In this work, the behaviour of two evolutionary algorithms, PSO and DE, on the simplest manifestations of the framework, ONECONE and TWOCONES geometries, and relate algorithm performance to a downhill walker (DHW) reference algorithm was analysed. The success rate, defined as the probability of optimal basin attainment, and inter-basin mobility were studied. It was found that local PSO is the slowest optimiser on the unimodal ONECONE but surpasses global PSO, DE and DHW algorithms on the bimodal TWOCONES in all problem instances below 70 dimensions.

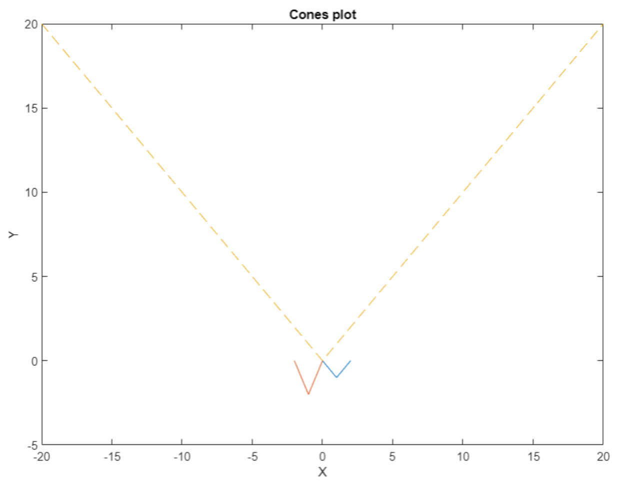

Figure 8: Example of a 2-CONES function

Future research perspectives

N cones

Based on the developed optimizer benchmark, a future general benchmark that includes N different cones and M funnels will be developed. This benchmark will allow constructing any problem with any complexity for testing and tuning EAs.

Parameter tuning methodology

A novel parameter tuning methodology might be needed to improve the generalization of evolutionary algorithms performance for different problems. As current methods for tuning the different parameters of these algorithms does not allow proper generalization to other problems, this novel methodology holds great potential for eliminating the need to retune parameters for each problem and therefore making the EA more practical for the real-world problems.

SwarmWalker algorithm

A novel algorithm that combines the simplicity of the DHW with the ability and the advantages of swarms is in the development and testing phase. It holds great potential for a highly efficient algorithm with fast operation.

Summary

The methods and systems described in this chapter provide several examples for the practical use of EA and metaheuristics. The examples include optimizers and optimization tools for geometric problems, multi-sensory systems and supply network security methods with the ability to solve complex problems and allocate agents to tasks in real time.
The structure of presented algorithms and protocols can be used for different scenarios and problems, but the parameters should be adjusted and tuned for a specific case study and be optimized based on the objective of the specific system for optimal results.
It enables designers to determine the best method for the specific problem which is essential for systems that require continuous operation, e.g., monitoring systems, lifesaving systems, complex systems with different types of agents and complex formations.

The main aspects covered in this chapter:

1. Evolutionary algorithms and meta-heuristics descriptions for systems collaboration, integration and optimization.
2. Optimization of some geometric problems and functions.
3. Supply network security framework, applicable to physical and digital systems which computes the expected value of system performance given the sensor, environmental, and task parameters.
4. A system applying heterogeneous agent algorithm to decide in real-time which sensor addresses which target without knowing a-priori when, where and which task will enter or leave the environment.
5. A suite of task administration protocols to handle risks and problems within the sensory system such as sensor availability, conflicts in task priorities and high time-consuming tasks.
6. A dual-layer system to ensure fault-tolerant sensor operation that is robust enough to sensor failure, deception, overloading and tampering.

Bibliography

Dorigo, M., 1992. *Optimization, learning and natural algorithms,* Milan: Ph.D. Thesis, Politecnico di Milano.

Fogel, L. J., 1963. *Biotechnology: Concepts and Applications.* New York: Prentice Hall .

Holland, J., 1975. *Adaptation in Natural and Artificial Systems.* s.l.:MIT press.

Karaboga, D., 2010. *Artificial bee colony algorithm,* s.l.: Scholarpedia.

Kennedy, J. & Eberhart, R., 1995. *Particle swarm optimization.* s.l., s.n., p. 1942–1948.

Kirkpatrick, S. C., Jr, D. G. & Vecchi., M. P., 1983. Optimization by simulated annealing. *Science,* pp. 671-680.

Kirkpatrick, S., Gelatt, J. C. D. & Vecchi, M. P., 1983. Optimization by Simulated Annealing. *Science,* pp. 671-680.

Koza, J., 1990. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems,* s.l.: Stanford University Computer Science Department technical report.

Pham, D. et al., 2005. *The Bees Algorithm,* Cardiff University, UK: Technical Note, Manufacturing Engineering Centre.

Rechenberg, I., 1973. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (PhD thesis),* s.l.: Frommann-Holzboog.

Schwefel, H.-P., 1974. *Numerische Optimierung von Computer-Modellen (PhD thesis),* s.l.: s.n.

Shi, Y. & Eberhart, R., 1998. *A modified particle swarm optimizer.* s.l., s.n., p. 69–73.

Storn, R. & Price, K., 1997. Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization,* p. 341–359.

Tkach, I. & Amador, S., 2021. Towards addressing dynamic multi-agent task allocation in law enforcement. *Autonomous Agents and Multi-Agent Systems.*

Tkach, I. & Blackweel, T., 2022. *Measuring Optimiser Performance on a Conical Barrier Tree.* Boston, s.n.

Tkach, I. & Edan, Y., 2020. *Distributed Heterogeneous Multi Sensor Task Allocation Systems.* s.l.:Springer Nature.

Tkach, I., Edan, Y., Jevtic, A. & Nof, S. Y., 2013. *Automatic Multi-sensor Task Allocation Using Modified Distributed Bees Algorithm.* s.l., IEEE International Conference on Systems, Man, and Cybernetics.

Tkach, I., Jevtic, A., Nof, S. Y. & Edan, Y., 2018. A Modified Distributed Bees Algorithm for Multi-Sensor Task Allocation. *Sensors,* p. 759.

Tkach, I., Nof, S. Y. & Edan, Y., 2017. Multi-sensor task allocation framework for supply networks security using task administration protocols. *International Journal of Production Research.*