SoftwareX 11 (2020) 100400

Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

Isula: A java framework for ant colony algorithms

Carlos Gavidia-Calderon [a,*], César Beltrán Castañon [b]

[a] University College London, Department of Computer Science, Gower Street, London, UK
[b] Pontificia Universidad Catolica del Peru, Department of Engineering, Av. Universitaria 1801 San Miguel, Lima, Peru

ARTICLE INFO

Article history:
Received 5 March 2019
Received in revised form 25 November 2019
Accepted 10 January 2020

Keywords:
Ant colony optimisation
Java
Travelling salesman problem
Image segmentation

ABSTRACT

Ant Colony Optimisation (ACO) algorithms emulate the foraging behaviour of ants to solve optimisation problems. They have proven effective in both academic and industrial settings. ACO algorithms share many features among them. Isula encapsulates these commonalities and exposes them for reuse in the form of a Java library. In this paper, we use the travelling salesman problem and image segmentation to showcase the framework capabilities using three top-performing ACO algorithms implemented in Isula. This framework is an open-source project available at GitHub, where is currently the most popular ACO java repository.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_61
Legal Code Licence	MIT Licence
Code versioning system used	git
Software code languages, tools, and services used	Java
Compilation requirements, operating environments & dependencies	Apache Maven
If available Link to developer documentation/manual	http://cptanalatriste.github.io/isula/doc/
Support email for questions	carlos.gavidia@pucp.edu.pe

1. Motivation and significance

Ant Colony Optimisation (ACO) algorithms, proposed by Dorigo et al. [1], solve optimisation problems by emulating the behaviour of ants in nature. When ants traverse a territory searching for food, they mark their path with pheromone. After they have located food, they make several trips from the food source to the nest, increasing the intensity of their path's pheromone trail. In case several ants have located the same food source and are also transporting food to the nest, the pheromone trail of the ant with the shortest path is more intense. The shortest path, in the same amount of time, has more two-way trips than longer paths that require more time for their traversal. Fellow ants are sensitive to pheromone and tend to select the path with the most intense pheromone trail. This behaviour also increases the pheromone intensity of the shortest path. Over time, the whole colony converges towards the optimal solution.

ACO algorithms are well-suited for generating high-quality solutions for computationally expensive problems. They have been successfully applied to a wide array of domains. Let us take path planning, an NP-complete problem, as an example. ACO algorithms have proven effective in producing short and collision-free routes [2,3], even in dynamic environments [4]. ACO algorithms are not limited to academic research: practitioners are using them in real-world scenarios – like industry and telecommunications – to solve optimisation problems [5].

Ant System (AS) is the first ACO algorithm proposed [1]. Researchers later developed new algorithms – also inspired by ant behaviour – either by improving, extending or adapting AS. Although these algorithms differ, given their common inspiration they have several commonalities. We rely on this reutilisation potential to offer a software framework – called Isula[1] – for quick implementation of ant colony algorithms [6]. We choose Java as programming language since is the most popular programming language at the time of this publication [7]. Isula includes several

* Corresponding author.
E-mail addresses: carlos.gavidia.15@ucl.ac.uk (C. Gavidia-Calderon), cbeltran@pucp.pe (C. Beltrán Castañon).

[1] The *Paraponera clavata* species of ant is called "isula" in Peru.

https://doi.org/10.1016/j.softx.2020.100400

Table 1
Daemon actions and policies included in Isula.

#	Isula type	Summary
1	*Offline Pheromone Update (AS)*	After each ant has built a solution, it deposits the corresponding pheromone increment to each solution component.
2	*Perform Evaporation (AS)*	Applies the evaporation ratio, hence reducing the pheromone amount in all the elements of the pheromone matrix.
3	*Random Node Selection (AS)*	Dictates how ants select the components to add to their current solution. The ant selects the component at random, where the probability of selection is a function of the pheromone value of the component and its heuristic information.
4	*Online Pheromone Update (ACS)*	While `antsystem.OfflinePheromoneUpdate` updates the pheromone values when the whole colony has finished building solutions, this ant policy updates the value as soon as individual ants finish constructing a candidate solution.
5	*Pseudo-Random Node Selection (ACS)*	While adding a component to a solution, the ant faces two possible options which it chooses at random: adopt `antsystem.RandomNodeSelection` or select the node with best values regarding pheromone and heuristic information.
6	*Start Pheromone Matrix (MMAS)*	MMAS keeps pheromone values between a range, restricting the evaporation and deposit of pheromone. At the beginning of the algorithm, this daemon action sets pheromones values to the maximum value.
7	*Update Pheromone Matrix (MMAS)*	A MMAS implementation only allows the best performing ant to deposit pheromone at the end of the iteration.

building blocks of ACO algorithms, which can be used out-of-the box or adapted in case the problem requires it. Among the building blocks available in the framework, we can mention node selection policies, pheromone update strategies, and ant colony initialisation procedures. Table 1 contains a subset of them. Isula currently supports the three top performing ACO algorithms [5]: Ant System (AS), Ant Colony Systems (ACS) and Max–Min Ant System (MMAS). We envision two use cases for Isula: (1) use Isula implementations of ACO algorithms out-of-the box, and (2) Built new ACO algorithms by reusing and adapting components already available in the framework.

Section 2 illustrates the first use case. The Travelling Salesman Problem (TSP) is usually used to showcase ACO algorithms [8]. We approach it to illustrate the framework internals by solving a TSP instance using AS. We also show how to reutilise the code we built for AS for a quick implementation of ACS for TSP.

Section 3 shows the second use case. We describe how to develop a medical image segmentation method by composing and adapting Isula components. We segment a brain MRI (magnetic resonance image) by combining two ant colony algorithms: an image thresholding algorithm [9] and an image segmentation one [10].

Our contribution follows:

- An introduction to Isula, a java framework for ACO algorithms (Section 2.1).
- A detailed description of how to use Isula's components out-of-the box to solve the travelling salesman problem using two well-established ACO algorithms (Ant System and Ant Colony System in Section 2.2).
- A complete example of how to use Isula to develop new algorithms by composing and adapting algorithm components. We implemented an brain MRI segmentation method (Section 3).

2. Software description

In this section, we start by giving a high-level overview of the main characteristics of the algorithms in the ACO metaheuristic, to later use the Travelling Salesman Problem to illustrate how to implement Java ACO algorithms using the Isula framework.

2.1. Implementing ACO algorithms with isula

Dorigo et al. proposed the ACO metaheuristic as an algorithmic framework for ACO algorithms, who would be then instances of this framework [11]. The algorithms that belong to the metaheuristic follow the structure depicted in Algorithm 1, taken from the book from Dorigo [8].

Algorithm 1 The ACO metaheuristic is pseudo-code

```
1: procedure ACOMetaheuristic
2:     procedure ScheduleActivities
3:         ConstructAntSolutions()
4:         UpdatePheromones()
5:         DaemonActions()                ▷ Optional
```

The *ScheduleActivities* function coordinates algorithm execution. During an iteration, the artificial ants traverse the problem graph and build a candidate solution each. While traversing the problem graph, ants randomly select which solution component to add to the candidate solution under construction. Depending on the algorithm, solution components can correspond to nodes or edges of the problem graph. The transition probability of each solution component guides this random selection process. This probability is a function the amount of pheromone on the solution component and a domain-dependent metric, also called heuristic information. The *ConstructAntSolutions* function represents the solution building process in Algorithm 1. The solution building process impacts the amount of pheromone available in each solution component. Ants deposit pheromone on the components of their candidate solution. Also, pheromone evaporation can be part of ACO algorithms to avoid rapid convergence and favour graph exploration. The *UpdatePheromones* function controls the pheromone update process. Finally, the *DaemonActions* functions implement behaviours on a global scale, instead of at ant-level.

Isula's goal is to accelerate the programming of instances of the ACO metaheuristic. It is a Java library that contains working implementations of the most common components of ACO algorithms. We show how to use Isula to solve optimisation problems in Fig. 1. It involves four steps:

Environment Definition Define a problem context by extending the `Environment` class.

Artificial Ant Definition Extend the `Ant` class to provide information necessary for building solutions.

Algorithm Configuration Configure an `AcoProblemSolver` instance according to the algorithm to use. This includes

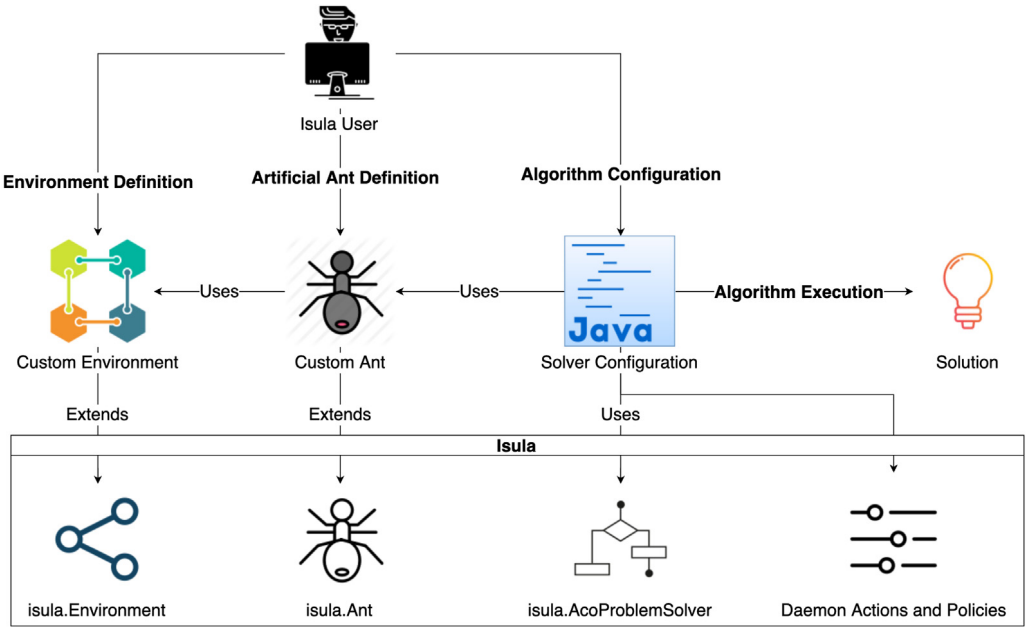

Fig. 1. Solving optimisation problems with Isula. This process has four steps: environment definition, artificial ant definition, algorithm configuration and algorithm execution.

daemon actions, node selection policies, and parameters like number of ants or evaporation rate. We show a subset of the types available at Isula in Table 1.

Algorithm Execution Call the `AcoProblemSolver.solveProblem()` method to obtain a result.

In the next subsection, we take a deep look at Isula internals and see how researchers and practitioners can use it to solve combinatorial optimisation problems.

2.2. Solving the Travelling Salesman Problem

Let us explore how to approach optimisation problems with Isula by solving an instance of the Travelling Salesman Problem (TSP). Given a list of cities the salesman needs to visit, we want to know in what order to visit them – i.e obtain a permutation of the cities – in order to minimise the distance travelled.

Environment and artificial ant definition. The first two steps for Isula problem solving require the definition of the environment and artificial ants. In the case of TSP, the environment would contain a matrix-like data structure for the storage of distance between cities, as well as its corresponding pheromone matrix. An artificial ant for solving TSP would need information about the distance travelled so far, and if it has already traversed all cities. For TSP, we developed `AntForTsp` and `TspEnvironment`, extending the abstract classes `Ant` and `Environment` available in the framework. These classes are available in GitHub at https://github.com/cptanalatriste/aco-tsp, along with all the code contained in this section.

Algorithm configuration. Listing 1 address the algorithm configuration step using the Ant System (AS) algorithm [1] to solve the berlin52 problem of the TSPLIB library [12].

Listing 1: Solving the Travelling Salesmen Problem, using an Isula-based implementation of Ant System.

```
// AcoTspWithIsula.java

double[][] problem = getRepresentationFromFile(fileName);

TspProblemConfiguration config = new
     TspProblemConfiguration(problem);
AntColony<Integer, TspEnvironment> colony =
     getAntColony(config);
TspEnvironment env = new TspEnvironment(problem);

AcoProblemSolver<Integer, TspEnvironment> solver = new
     AcoProblemSolver<>();
solver.initialize(env, colony, config);
solver.addDaemonActions(new StartPheromoneMatrix<Integer,
     TspEnvironment>(),
       new PerformEvaporation<Integer, TspEnvironment>());

solver.addDaemonActions(getPheromoneUpdatePolicy());

solver.getAntColony().addAntPolicies(new
     RandomNodeSelection<Integer, TspEnvironment>());
solver.solveProblem();
```

The `TspProblemConfiguration` class implements the `ConfigurationProvider` interface from Isula. `ConfigurationProvider` instances contain the parameters of the ACO algorithm to execute, like the number of ants, pheromone evaporation ratio and the number of iterations. The `TspEnvironment` class extends the `Environment` class from the framework, which represents the environment the ants traverse while building solutions. It also stores the pheromone matrix that contains the pheromone value per solution component. To use AS we need to provide our `AcoProblemSolver` instance with the appropriate daemon actions and ant policies. AS was the first ACO algorithm proposed, and its main feature is that *all* the ants that build a solution after an iteration have the right to perform pheromone deposits. Other ACO algorithms prefer to restrict this behaviour. We rely on the types `antsystem.OfflinePheromoneUpdate` (`getPheromoneUpdatePolicy()` returns an instance of this class), `antsystem.PerformEvaporation` and `antsystem.StartPheromoneMatrix` to update pheromone values according to AS requirements. Table 1 contains a description of each of these classes.

The ants in the colony are the ones in charge of making candidate solutions. Hence, a key component in algorithm implementation with Isula is constructing the ant colony and its members.

In Listing 1, the `getAntColony()` method produces an instance of the `AntColony` class of the framework, but overriding the `createAnt()` method to produce instances of the `AntForTsp` class. This class is tailored for TSP solution construction. We have also added the `antsystem.RandomNodeSelection` policy to all the ants in the colony. This policy guides the criteria the ants use to add components to their candidate solution. Listing 2, available in Github at https://github.com/cptanalatriste/isula , is a snippet from this class.

Listing 2: Obtaining transition probabilities in Isula using a Random Node Selection policy.

```
// RandomNodeSelection.java

private Double getHeuristicTimesPheromone(E env,
    ConfigurationProvider config, C component) {

    Double heuristicValue =
         getAnt().getHeuristicValue(component,
           getAnt().getCurrentIndex(), env);
    Double pheromoneTrailValue =
         getAnt().getPheromoneTrailValue(component,
           getAnt().getCurrentIndex(), env);

    if (heuristicValue == null || pheromoneTrailValue == null) {
        throw new SolutionConstructionException();
    }

    return Math.pow(heuristicValue,
         config.getHeuristicImportance()) *
        Math.pow(pheromoneTrailValue,
             config.getPheromoneImportance());
}
```

As mentioned in Section 2.1, ants select components for their solutions randomly, with a transition probability dependant on pheromone values and heuristic information. `antsystem.RandomNodeSelection` obtains this probability for all *valid* solution components and randomly selects one according to the transition probability values. This class is a part of the Isula framework and is ready to use without modifications in the case of TSP. The `AntForTsp` class is tailored to build solutions for the TSP. Hence, its `getHeuristicValue()` method returns the distance of a potential solution component, with respect to the ant's position in the problem graph. When using Isula to solve a novel optimisation problem, framework users should extend the base `Ant` class and provide an adequate implementation of `getHeuristicValue()` to guide the solution construction process.

Algorithm execution. Lets now address the algorithm execution step to look in detail how Isula approaches combinatorial optimisation. Fig. 2 is a sequence diagram of the `AcoProblemSolver.solveProblem()` method, which is the Isula's implementation of the procedure described in Algorithm 1. Isula distributes most of the *ScheduleActivities* responsibilities between the `AcoProblemSolver` class and the `AntColony` class. The `AcoProblemSolver` class delegates the solution building responsibilities to the `AntColony` class, while it keeps track of the best solution produced so far. The `AcoProblemSolver` class is also in charge of triggering *DaemonActions*, that can happen either before or after the *ConstructAntSolutions* phase, implemented by the `AntColony.buildSolutions()` method. Users of the framework can instantiate the `AcoProblemSolver` class without modifications, as it should support most of the ACO algorithms. If is not the case, Isula supports its extension.

In summary, in order to solve a combinatorial optimisation problem Isula requires its users to implement three types: (1) A `Environment` subclass for representing the problem context, (2) An `Ant` subclass for the ants that traverse the environment and build candidate solutions, and (3) a `ConfigurationProvider` implementation containing the algorithm parameters. The Isula framework handles the rest.

Code reuse opportunities. Now that we solved TSP using an Isula implementation, we can reuse the code we developed to use other ACO algorithms to solve TSP problems. We will use Ant Colony System (ACS) and Ant System (AS) to showcase the reuse potential between ACO algorithms and their Isula implementations. ACS changes the way pheromone updates: It happens both when ants are building solutions (called *local* pheromone update) and when the whole colony has finished (called *global* pheromone update). Also, ants in an ACS algorithm use a pseudo-random node selection policy, unlike the random node selection policy used in AS. Listing 3, available in Github at https://github.com/cptanalatriste/aco-acs-tsp, shows how to solve a TSP using Ant Colony System (ACS) [13], reusing the types we developed while applying AS.

Listing 3: Solving the Travelling Salesmen Problem, using an Isula-based implementation of Ant Colony System.

```
// AcoAcsTspWithIsula.java

double[][] problem =
     AcoTspWithIsula.getRepresentationFromFile(fileName);

AcsTspProblemConfiguration config = new
     AcsTspProblemConfiguration(problem);
AntColony<Integer, TspEnvironment> colony =
     AcoTspWithIsula.getAntColony(config);
TspEnvironment env = new TspEnvironment(problem);

AcoProblemSolver<Integer, TspEnvironment> solver = new
     AcoProblemSolver<>();
solver.initialize(env, colony, config);
solver.addDaemonActions(new StartPheromoneMatrix<Integer,
     TspEnvironment>());
solver.addDaemonActions(getGlobalPheromoneUpdatePolicy());

solver.getAntColony().addAntPolicies(
    getLocalPheromoneUpdatePolicy());
solver.getAntColony().addAntPolicies(new
     PseudoRandomNodeSelection<Integer, TspEnvironment>());
solver.solveProblem();
```

We can use the `AntForTsp` and `TspEnvironment` classes without modifications. We can also reuse the `antsystem.OfflinePheromoneUpdate` and `antsystem.StartPheromoneMatrix` during pheromone update process. Due to the nature of the ACS algorithm, we need to include two additional policies already present in the Isula framework: `acs.PseudoRandomNodeSelection` and `antsystem.OnlinePheromoneUpdate`. We detail the behaviour of these classes in Table 1.

3. Illustrative example

In the previous section, we applied well-known ant colony algorithms to a well-known problem. Here, we implement an ant colony algorithm for the medical imaging domain, proposed by us in previous work [14]. As shown in Fig. 3, our method takes as input a magnetic resonance (MR) image of a brain (like Fig. 5 from the BrainWeb [15]) to produce three segments from it: grey matter, white matter and cerebrospinal fluid (CF). Fig. 4 contains the grey matter segment our method extracted from the MR brain image in Fig. 5. This segmentation task is useful for computer-assisted surgery, anomaly detection and the quantification of sclerosis lesions [16].

Our method is a composition of two ACO algorithms. After image pre-processing, the *ACO image thresholding algorithm* extracts the cerebrum – our region of interest – and excludes the

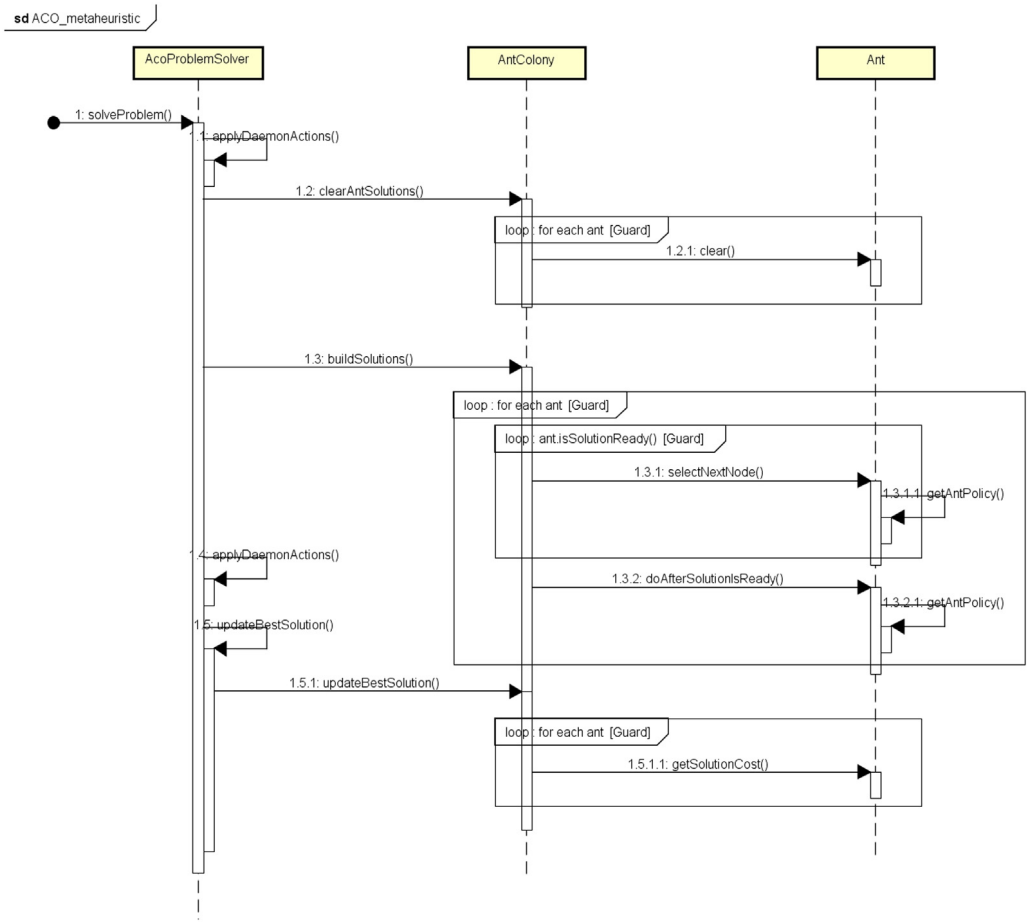

Fig. 2. Sequence diagram of the *AcoProblemSolver.solveProblem()* method.

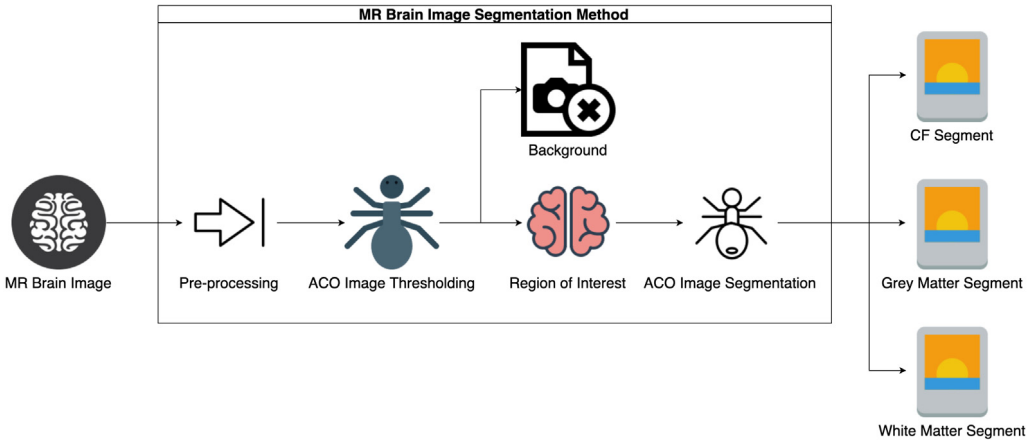

Fig. 3. MR Brain image segmentation method based on ACO algorithms [14].

pixels corresponding to skull, muscle and fat. Then, the cerebrum pixels are then processed by the *ACO image segmentation algorithm*, which produces images corresponding to each relevant segment. The image segmentation algorithm is resource intensive, so it benefits greatly from the reduced input produced by image thresholding. The implementation of our method in Isula is available at GitHub (https://github.com/cptanalatriste/aco-image-segmentation), including image pre-processing. In this section, we do not cover image pre-processing and focus only on the two ACO algorithm components.

ACO Image thresholding. Malisia and Tizhoosh proposed the ACO algorithm we selected for thresholding [9]. They based their algorithm on Ant System, discussed in Section 2. The algorithm divides the input image in two regions: region of interest – the cerebrum – and background. To accomplish this, they assign artificial ants to each image pixel, making them seek low-greyscale pixel values. While traversing the image they deposit pheromone in each pixel. Pixel pheromone is used as feature for the clustering algorithm that produce the final segments. We show the image thresholding algorithm configuration code in Listing 4. This code

Fig. 4. Grey matter segment extracted from Fig. 5 using our proposed ACO algorithm.

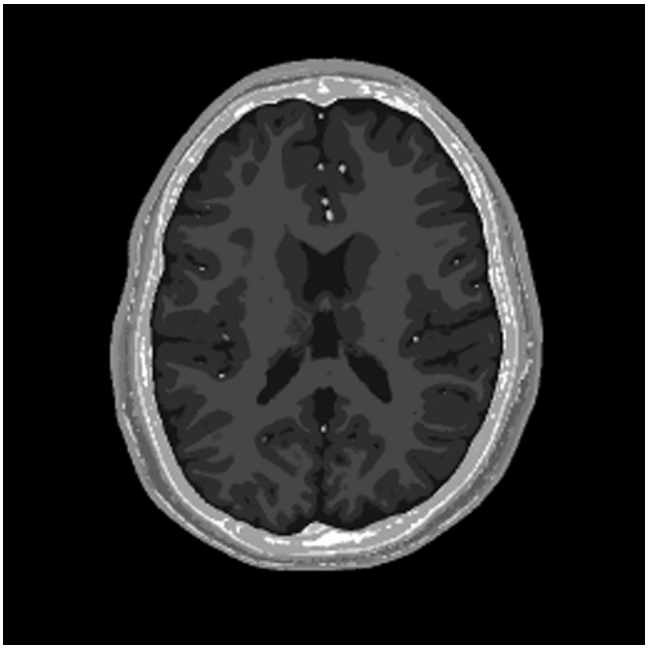

Fig. 5. An axial MR brain image, obtained from the BrainWeb database [15].

is also available in GitHub at https://github.com/cptanalatriste/aco-image-thresholding:

Listing 4: Implementing image thresholding, using an Isula-based implementation of Ant System.

```
// AcoImageThresholding.java

ConfigurationProvider config = new ProblemConfiguration();
AcoProblemSolver<ImagePixel> solver = new
     AcoProblemSolver<ImagePixel>();
EnvironmentForImageThresholding env = new
     EnvironmentForImageThresholding(
    imageGraph, ProblemConfiguration.NUMBER_OF_STEPS);

ImageThresholdingAntColony colony = new
     ImageThresholdingAntColony();
colony.buildColony(env);

solver.setConfigurationProvider(config);
solver.setEnvironment(env);
solver.setAntColony(colony);

solver.addDaemonActions(new
     StartPheromoneMatrix<ImagePixel>(),
    new RandomizeHive(), new PerformEvaporation<ImagePixel>());
colony.addAntPolicies(new NodeSelectionForImageThresholding(),
    new OnlinePheromoneUpdateForThresholding());

solver.solveProblem();
```

We reutilise Isula types like `ConfigurationProvider` and `AcoProblemSolver`. Also, since the algorithm is an extension of AS we can also rely on the `antsystem.PerformEvaporation` and `antsystem.StartPheromoneMatrix` daemon actions. Due to the new domain, we require further customisation. We designed Isula with these scenarios in mind, and it supports its extension and modification. For image thresholding with AS, we developed a specialised ant colony type (`ImageThresholdingAntColony`). We also adapted the pseudo-random node selection (`NodeSelectionForImageThresholding`) and online pheromone updated policies available in Isula (`OnlinePheromoneUpdateForThresholding`). Since we based these extensions on existing Isula implementations, only the relevant methods were over-written. The algorithm requires to place ants at random pixels at the beginning of each iteration. We developed the `RandomizeHive` daemon action to accomplish this.

ACO Image segmentation. Once the image thresholding algorithm extracts the cerebrum, the image segmentation algorithm can start processing it. We implemented the algorithm proposed by Ouadfel and Batouche [10]. In their approach, each artificial ant is in charge of building a partition. They based their algorithm on Max–Min Ant System (MMAS) [17] – another classic ACO algorithm – so we can rely on Isula types for its implementation. In MMAS, only the best performing ant can deposit pheromone at the end of the iteration. Also, the pheromone values per solution component must belong to a specific range. Listing 5 shows how to configure the solver:

Listing 5: Implementing image segmentation, using an Isula-based implementation of Max–Min Ant System.

```
// AcoImageSegmentation.java

ConfigurationProvider config = ProblemConfiguration
       .getInstance();
int numberOfClusters =
     ProblemConfiguration.getInstance().getNumberOfClusters()
EnvironmentForImageSegmentation env = new
     EnvironmentForImageSegmentation(
    imageGraph, numberOfClusters);

ImageSegmentationAntColony antColony = new
     ImageSegmentationAntColony(
    config.getNumberOfAnts(), env.getNumberOfClusters());
antColony.buildColony(env);

AcoProblemSolver solver = new
     AcoProblemSolver<ClusteredPixel>();

solver.setConfigurationProvider(config);
solver.setEnvironment(env);
solver.setAntColony(antColony);
```

Table 2
Isula-based implementations of ACO algorithms at GitHub.

#	Problem and ACO Algorithm	GitHub Project
1	Travelling Salesman with Ant System (Brownle [22])	cptanalatriste/aco-tsp
2	Travelling Salesman with Ant Colony System (Brownlee [22])	cptanalatriste/aco-acs-tsp
3	Flow-Shop Scheduling with Max–Min Ant System (Stützle [23])	cptanalatriste/aco-flowshop
4	Image Thresholding with Ant System (Malisia and Tizhoosh [9])	cptanalatriste/aco-tsp
5	Image Clustering with Max–Min Ant System (Ouadfel and Batouche [22])	cptanalatriste/aco-tsp

```
solver.addDaemonActions(
    new StartPheromoneMatrixForMaxMin<ClusteredPixel>(),
    new ImageSegmentationUpdatePheromoneMatrix());
antColony.addAntPolicies(new
     ImageSegmentationNodeSelection());

solver.solveProblem();
ClusteredPixel[] bestPartition = solver.getBestSolution();
```

The complete implementation is available in GitHub at https://github.com/cptanalatriste/aco-image-segmentation. Like with the thresholding algorithm, the segmentation algorithm also uses the pseudo-random node selection policy from ACS, so `ImageSegmentationNodeSelection` extends `acs.PseudoRandomNodeSelection`. The pheromone update process follows MMAS rules, so we use the Isula types `maxmin.StartPheromoneMatrixForMaxMin` and `maxmin.UpdatePheromoneMatrixForMaxMin` (extended by `ImageSegmentationUpdatePheromoneMatrix`). We describe these types in Table 1. Like in the previous examples, we developed subclasses of `Environment`, `Ant` and `AntColony`, adapted to the problem domain. This custom code can be used along with Isula's built-in types.

4. Impact

ACO algorithms are not restricted to the academic domain: they are actively used to solve optimisation problems at industry [5]. Being Java the most popular programming language [7], there is a need for a robust java library for quick ACO implementations. The development of Isula started as part of one of the author's Master dissertation in 2015 [14]. At the time of publication, Isula has been used for implementing ACO algorithms for image segmentation [18] and road extraction [19].

Isula is an open source project at GitHub, the world's largest coding platform. Table 2 contains the Isula implementations of ACO algorithms we have developed so far. We built these open-source projects to showcase Isula capabilities, so the code follows a tutorial-style and its fully documented. At the time of this publication, Isula is the *most popular* ant colony optimisation project in Java at GitHub, in both forks and stars, among 24 code repositories in the area [20]. Without considering programming language, is the third most popular project [21] among 98 code repositories .

Thanks to the feedback received over the years we have greatly improved Isula's tutorials and documentation. We hope to continue enabling the practitioners and research community on the implementation of ACO algorithms. As seen in this paper, the framework is flexible enough to support several problem domains and different ACO algorithms.

4.1. Isula roadmap

In this subsection, we present the extensions of the Isula framework we have planned.

Multi-objective optimisation. The ACO metaheuristic (Section 2.1) was originally designed to support *single-objective* combinatorial optimisation problems [24]. The current version of Isula has the same limitation: it does not provide multi-objective support out-of-the-box. Researchers have later developed ant-inspired algorithms that can produce a set of Pareto-optimal solutions for a given multi-objective optimisation problem [25]. The architecture of such algorithms differs the original ACO metaheuristic proposal, requiring multiple pheromone matrices, multiple colonies and multi-dimensional heuristic information. López-Ibáñez and Stützle proposed MOACO as an algorithmic framework for multi-objective ACO algorithms [24]. As part of our roadmap, we plan for Isula to support MOACO algorithms in the future.

Parallel ACO. Our roadmap also includes extending Isula to support parallel implementations of ACO algorithms. Current proposals require the concurrent execution of ant sub-colonies over multiple processors, along with a policy for information exchange between them [25]. This information can be the best solution found by each sub-colony, or even the entirety of the pheromone matrix [26]. The current version of Isula does not support these architectures, but we expect a future release addressing these concerns.

Parameter optimisation. The performance of ACO algorithms is highly dependent of their parameter values [27]. The current version of Isula, as most ACO algorithms in the literature, keep parameter values constant during algorithm execution. A new generation of ACO algorithms are able to adapt parameter values at runtime [28–30]. These algorithms can be grouped in two categories: Those that schedule parameter update before algorithm execution, and those that update parameters according to the current search state. We planned to extend Isula to support these parameter update strategies.

5. Conclusion

The ACO metaheuristic has proven effective in solving optimisation problems. The instances of this algorithmic framework share many features: Isula relies on this to offer ready-to-use building blocks in the form of a robust Java library. This way, the implementation of new ACO algorithms is faster. Isula supports multiple domains and ACO algorithms, and is easily extensible as shown in this paper. It is currently the most popular GitHub project for ant colony optimisation in Java, and we hope more researchers and practitioners adopt Isula and help with its improvement.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B 1996;26(1):29–41. http://dx.doi.org/10.1109/3477.484436.
[2] García MAP, Montiel O, Castillo O, Sepúlveda R, Melin P. Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. Appl Soft Comput 2009;9(3):1102–10. http://dx.doi.org/10.1016/j.asoc.2009.02.014.
[3] Ross OM, Sepúlveda R, Castillo O, Melin P. Ant colony test center for planning autonomous mobile robot navigation. Comput Appl Eng Educ 2013;21(2):214–29. http://dx.doi.org/10.1002/cae.20463.
[4] Brand M, Masuda M, Wehner N, Yu X-H. Ant colony optimization algorithm for robot path planning. In: 2010 international conference on computer design and applications, vol. 3. 2010. p. V3–436–V3–440.

[5] Dorigo M, Birattari M, Stutzle T. Ant colony optimization. IEEE Comput Intell Mag 2006;1(4):28–39. http://dx.doi.org/10.1109/MCI.2006.329691.
[6] Isula: A framework for ant colony algorithms. http://cptanalatriste.github.io/isula/. [Accessed 17 February 2019].
[7] TIOBE Index for February 2019. https://www.tiobe.com/tiobe-index/. [Accessed 24 February 2019].
[8] Dorigo M, de Recherches Du Fnrs Marco Dorigo D, Stützle T, Stützle T. Ant colony optimization. A Bradford book, BRADFORD BOOK; 2004.
[9] Malisia AR, Tizhoosh HR. Applying ant colony optimization to binary thresholding. In: Proceedings of the international conference on image processing. IEEE; 2006, p. 2409–12. http://dx.doi.org/10.1109/ICIP.2006.312948.
[10] Ouadfel S, Batouche M. Unsupervised image segmentation using a colony of cooperating ants. In: Bülthoff HH, Lee S, Poggio TA, Wallraven C, editors. Biologically motivated computer vision second international workshop, proceedings. Lecture notes in computer science, vol. 2525, Springer; 2002, p. 109–16. http://dx.doi.org/10.1007/3-540-36181-2_11.
[11] Dorigo M, Caro GD, Gambardella LM. Ant algorithms for discrete optimization. Artif Life 1999;5(2):137–72. http://dx.doi.org/10.1162/106454699568728.
[12] TSPLIB. https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ [Accessed 17 February 2019].
[13] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1997;1(1):53–66. http://dx.doi.org/10.1109/4235.585892.
[14] Gavidia-Calderon CG. Segmentación de Imágenes Médicas Mediante Algoritmos de Colonia de Hormigas [Master's thesis], Pontificia Universidad Católica del Perú, Escuela de Posgrado. Mención Magíster en Informática; 2014.
[15] BrainWeb Simulated Brain Database. http://brainweb.bic.mni.mcgill.ca/brainweb/. [Accessed 23 February 2019].
[16] Xu R, Luo L, Ohya J. Segmentation of brain MRI. In: Chaudhary V, editor. Advances in brain imaging. Rijeka: IntechOpen; 2012, http://dx.doi.org/10.5772/27596.
[17] Stutzle T, Hoos H. MAX-MIN ant system and local search for the traveling salesman problem. In: Proceedings of 1997 IEEE international conference on evolutionary computation. 1997, p. 309–14. http://dx.doi.org/10.1109/ICEC.1997.592327.
[18] Nadipally M. Chapter 2 - optimization of methods for image-texture segmentation using ant colony optimization. In: Hemanth DJ, Gupta D, Balas VE, editors. Intelligent Data Analysis for Biomedical Applications. Intelligent Data-Centric Systems, Academic Press; 2019, p. 21 – 47. http://dx.doi.org/10.1016/B978-0-12-815553-0.00002-1.
[19] Wang Y, Tang C, Yang J, Wei L. Road extraction from high-resolution remotely sensed image based on improved ant colony optimization method. In: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech 2018, Athens, Greece, August 12-15, 2018. 2018, p. 300–9. http://dx.doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00058.
[20] Ant colony optimization in Java at GitHub. https://github.com/topics/ant-colony-optimization?l=java. [Accessed 24 February 2019].
[21] Ant colony optimization at GitHub. https://github.com/topics/ant-colony-optimization. [Accessed 24 February 2019].
[22] Brownlee J. Clever algorithms : nature-inspired programming recipes. Lulu; 2011.
[23] Stützle T, et al. An ant approach to the flow shop problem. In: Proceedings of the 6th European congress on intelligent techniques and soft computing, vol. 3. 1998. p. 1560–64.
[24] López-Ibáñez M, Stützle T. The automatic design of multiobjective ant colony optimization algorithms. IEEE Trans Evol Comput 2012;16(6):861–75. http://dx.doi.org/10.1109/TEVC.2011.2182651.
[25] Dorigo M, Stützle T. Ant colony optimization: Overview and recent advances. In: Gendreau M, Potvin J-Y, editors. Handbook of metaheuristics. Springer International Publishing; 2019, p. 311–51. http://dx.doi.org/10.1007/978-3-319-91086-4_10.
[26] Twomey C, Stützle T, Dorigo M, Manfrin M, Birattari M. An analysis of communication policies for homogeneous multi-colony ACO algorithms. Inform Sci 2010;180(12):2390–404. http://dx.doi.org/10.1016/j.ins.2010.02.017.
[27] Stützle T, López-Ibáñez M, Pellegrini P, Maur M, de Oca MAM, Birattari M, Dorigo M. Parameter adaptation in ant colony optimization. In: Hamadi Y, Monfroy E, Saubion F, editors. Autonomous search. Springer; 2012, p. 191–215. http://dx.doi.org/10.1007/978-3-642-21434-9_8.
[28] Olivas F, Valdez F, Castillo O, González CI, Martinez GE, Melin P. Ant colony optimization with dynamic parameter adaptation based on interval type-2 fuzzy logic systems. Appl Soft Comput 2017;53:74–87. http://dx.doi.org/10.1016/j.asoc.2016.12.015.
[29] Castillo O, Neyoy H, Soria J, Melin P, Valdez F. A new approach for dynamic fuzzy logic parameter tuning in ant colony optimization and its application in fuzzy control of a mobile robot. Appl Soft Comput 2015;28:150–9. http://dx.doi.org/10.1016/j.asoc.2014.12.002.
[30] Castillo O, Lizárraga E, Soria J, Melin P, Valdez F. New approach using ant colony optimization with ant set partition for fuzzy control design applied to the ball and beam system. Inform Sci 2015;294:203–15. http://dx.doi.org/10.1016/j.ins.2014.09.040.