Injecting domain knowledge in multi-objective optimization problems: A semantic approach

Cristóbal Barba-González, Antonio J. Nebro, José García-Nieto, María del Mar Roldán-García, Ismael Navas-Delgado, José F. Aldana-Montes

Departamento de Lenguajes y Ciencias de la Computación, ITIS Software, University of Málaga, 29071 Málaga, Spain

Abstract

In the field of complex problem optimization with metaheuristics, semantics has been used for modeling different aspects, such as: problem characterization, parameters, decision-maker's preferences, or algorithms. However, there is a lack of approaches where ontologies are applied in a direct way into the optimization process, with the aim of enhancing it by allowing the systematic incorporation of additional domain knowledge. This is due to the high level of abstraction of ontologies, which makes them difficult to be mapped into the code implementing the problems and/or the specific operators of metaheuristics. In this paper, we present a strategy to inject domain knowledge (by reusing existing ontologies or creating a new one) into a problem implementation that will be optimized using a metaheuristic. Thus, this approach based on accepted ontologies enables building and exploiting complex computing systems in optimization problems. We describe a methodology to automatically induce user choices (taken from the ontology) into the problem implementations provided by the jMetal optimization framework. With the aim of illustrating our proposal, we focus on the urban domain. Concretely, we start from defining an ontology representing the domain semantics for a city (e.g., building, bridges, point of interest, routes, etc.) that allows defining a-priori preferences by a decision maker in a standard, reusable, and formal (logic-based) way. We validate our proposal with several instances of two use cases, consisting in bi-objective formulations of the Traveling Salesman Problem (TSP) and the Radio Network Design problem (RND), both in the context of an urban scenario. The results of the experiments conducted show how the semantic specification of domain constraints are effectively mapped into feasible solutions of the tackled TSP and RND scenarios. This proposal aims at representing a step forward towards the automatic modeling and adaptation of optimization problems guided by semantics, where the annotation of a human expert can be now considered during the optimization process.

Keywords: Multi-Objective Optimization, Decision Making, Metaheuristics, Domain Knowledge, Semantic Web Technologies, Ontology

1. Introduction

The optimization of problems consisting of two or more conflicting objectives has received plenty of attention over the last two decades. In these years, multi-objective optimization problems (MOPs) have been dealt with metaheuristics [1] with a high degree of success, leading to a large amount of algorithms that provide an accurate approximation to the Pareto front of the solved problem [2].

However, finding this front is only a part of the optimization process, which has to be followed by a multi-criteria decision making (MCDM) step, as the decision maker has to choose which solutions are the most adequate according to some criteria or preferences. This would be an a-posteriori MCDM approach, but preferences can also be indicated a-priori (before starting the optimization) or interactively (during the execution of the optimization algorithm) [3, 4, 5].

In computer and information sciences, an ontology defines a set of representational primitives for modeling a domain of knowledge, which is used to include knowledge about requirements statements in some domain [6], or discourse [7, 8]. The representational primitives are typically concepts (or classes), attributes (or properties), class members (class instances) and relationships (property instances). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application.

Ontologies have been applied to many domains, including health [9], traffic [10], engineering [11], etc., and the field of optimization is not an exception. The work presented in [12] attempts to explain the ability of an evolutionary algorithm to provide efficient solutions that typically depends on the specific type of problem regarding the most efficient search. Moreover, this study proposed the ECO ontology, which can be used for suggesting efficient strategies for solving problems through Evolutionary Computation. The ontology proposed in [13] is aimed at modeling and systematizing the knowledge

Email addresses: cbarba@lcc.uma.es (Cristóbal Barba-González), antonio@lcc.uma.es (Antonio J. Nebro), jnieto@lcc.uma.es (José García-Nieto), mmar@lcc.uma.es (María del Mar Roldán-García), ismael@lcc.uma.es (Ismael Navas-Delgado), jfam@lcc.uma.es (José F. Aldana-Montes)

of preference-based multi-objective evolutionary algorithms, and it can be used to find algorithms according to some properties (such as authors, whether the algorithm has been used to solve a given problem, or whether the algorithm is an extension of an existing one). Recently, the BIGOWL ontology [14] has been proposed to include the description of both, algorithms and problems, in the context of Big Data analytic workflows where multi-objective optimization, among other tasks, can be worked.

Compared with these works, our purpose is not to model multi-objective optimization entities or reasoning about them, but to consider ontologies as a mean of expressing problem domain knowledge [15], [16] and to inject this knowledge into the software implementation of a multi-objective problem, which will be optimized by metaheuristics. Thus, the decision makers' preferences or requirements (defined through an ontology) can be annotated and queried to be taken into account during the execution of the optimization algorithm. Consequently, the algorithm outcome reflects those preferences in the variable and objective spaces that are compared during the optimization process. The main goal is to automatically capture the domain knowledge during the process of multi-objective problem optimization, without needing to manually encode this knowledge, in the form of problem constraints, in the software implementation. This would be beneficial for the generation of advanced applications where optimization processes could be modeled in a semi-transparent way by an expert in the problem domain, without requiring strong expertise in optimization algorithms.

In this paper, we focus on the a-priori approach, following the idea of "evolutionary algorithm + domain knowledge = real world evolutionary computation" [17]. Our proposal combines ontologies defining the domain knowledge of a multi-objective optimization problem with the mapping of that knowledge as a-priori preferences in the context of a software library of metaheuristics. We define a methodology to enable the automatic mapping of user preferences (described through individuals of the classes and properties in the ontology) in the form of constraints in a multi-objective optimization problem definition. The implementation of our methodology has been carried out using the optimization framework jMetal [18][19].

To show the usability and potentials of our proposal, we focus on a specific domain of knowledge in the urban context of a real-world city. In order to describe all the concepts and their relationships in this context, we propose a domain ontology called Urban[1], which is used as an integration model of other sub-ontologies, namely: BIGOWL and smart city [20, 21, 22]. We apply our proposal to two case studies based on the bi-objective formulations of the Traveling Salesman Problem (TSP) [23] and the Radio Network Design problem (RND) [24]. The former belongs to the field of logistics and transportation, while the latter is an optimization problem found in telecommunications. These problems are enriched with contextual information such as VIP nodes (e.g., preferred nodes that have to be visited the first ones) or linked nodes (when node A is visited, the next node in the route must be node B) in the case of TSP, and antenna positions banned (e.g., because there is a school) in the case of RND problems.

All in all, the main contributions of this paper are as follows:

1. We provide a novel approach for automatically injecting domain knowledge, in form of user preferences, into the software code that implements the optimization problem and the metaheuristic used to tackle it. The conceptual relationships in the domain of knowledge that involve these preferences are mapped as problem constraints for guiding the search in the optimization process. To the best of our knowledge, it is a first attempt in this direction, which is supported with software code and use cases to validate the proof of concept.
2. For validation purposes, we have developed a new ontology, called Urban, devoted to the urban domain knowledge, which re-uses classes and properties from BIGOWL and (by means of alignments) with smart city ontologies including: km4city [20], geosparql [21], smart-city ontology [22], among others. Urban ontology allows defining a-priori user preferences in a reusable and formal way. Therefore, knowledge domain is represented independently of the optimization problem programming language, and can be reused by different implementations of the same algorithm.
3. The proposed use cases incorporate implementations of the traveling salesman problem (TSP) and the radio network design (RND) problem in jMetal, so that those preferences previously annotated by the decision maker through the domain ontology (Urban) can be incorporated during the optimization process of a given optimizer. In order to adapt this solution to a new context, the user only needs to update the domain ontology, so there is no need of changing the software implementation.
4. We have tested our approach with these two use cases with different types of preferences. Each experiment only requires updating the ontology instances to define these preferences. In particular, TSP uses preferences for preferred node, sub-route, node in the middle of the route, and node in the first quarter of the route. RND describes uses for antenna in a fix location, antenna location avoiding a building or setting a minimum desired coverage.

The results of the experimentation carried out show how the semantic specification of domain constraints are effectively mapped into feasible solutions of the TSP and RND scenarios. These constraints can be updated in the ontology, and hence, they are automatically considered in the problem implementation.

The rest of paper is organized as follows. Section 2 describes background concepts related to ontologies, multi-objective optimization, metaheuristics and the optimization framework jMetal. In Section 3, the methodology for designing our ontology is described. Section 4 describes our approach to inject the domain knowledge into the code implementation of a problem. Section 5 depicts the two use cases for validation, which

[1]Ontology available on `https://github.com/KhaosResearch/UrbanOntology`

is followed by discussions in Section 6. The conclusions and future research lines are included in Section 7. Finally, the examples are fully described using the OWL Manchester syntax [25] in the Appendix.

2. Background Concepts

In this section, we provide basic background about multi-objective optimization, including decision making and metaheuristics. Besides, we focus on the concept of ontology and related technologies, and we describe BIGOWL and other ontologies of the domain of smart cities, which are the basis of the strategy we propose.

2.1. Multi-objective Optimization and Decision Making

Multi-Objective optimization is a discipline focused on dealing with problems having multiple objective functions that have to be maximized or minimized, at the same time. Assuming minimization, without loss of generality, a multi-objective optimization problem can be formulated as:

$$\begin{aligned} \text{minimize} \quad & \{f_1(x), f_2(x), ..., f_m(x)\} \\ \text{subject to} \quad & g_j(x) \geq 0, \quad j = 1, 2, ..., n \\ & h_k(x) = 0, \quad k = 1, 2, ..., p \\ & x \in S \end{aligned} \tag{1}$$

with $m \geq 2$ conflicting objective functions $f_i : S \rightarrow \mathbb{R}$, x is a vector of decision variables from the feasible set S (search space), which is determined by n and p inequality and equality constraints, respectively.

As the objectives are conflicting among them, improving one implies the worsening of the others. As a consequence, the optimum of a multi-objective problem is not usually a single solution, but a set of non-dominated (trade-off) solutions called *Pareto set*. Two solutions are non-dominated when none of them is better than the others in all their corresponding function values. In this way, the solutions in the Pareto set are those that cannot be dominated by any other solution in the search space. A concept related to the Pareto set is the *Pareto front*, which is the correspondence of all the solutions in the Pareto set in the objective space.

Solving real-world multi-objective optimization problems with exact techniques is impractical [26] in many situations, so other kinds of algorithms are used instead. In particular, metaheuristics [1] are a family of non-exact algorithms that have become very popular, because they can provide quasi-optimal solutions in a limited budget of time and/or resources. Those algorithms are aimed at finding accurate approximations of the Pareto front of a given problem. An example is illustrated in Figure 1, where the gray square points represent the trade-off solutions found by an algorithm when solving a civil engineering problem where two conflicting objectives, the weight (f_1) and the deformation (f_2) of a structure, must be minimized.

Once an approximation to the Pareto front has been found, the next step is the multi-criteria decision making (MCDM). In this step, the expert in the problem domain (the decision maker) has to choose one or more solutions according to some preferences or criteria. This is the a-posteriori approach. The red circle points in Figure 1 represent three possible alternatives. For example, the solution in the bottom right of the front represents a heavy structure with a very low degree of deformation.

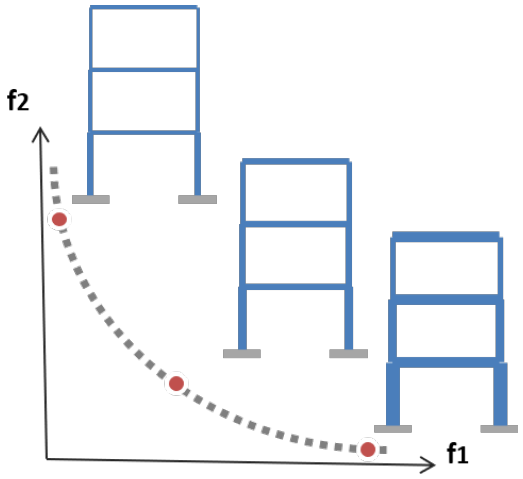

Figure 1: Example of Pareto front approximation.

An alternative approach for MCDM is to indicate the preferences a-priori (e.g., reference points) before starting the search. This is the strategy we consider in this work, and the basic idea is to take advantage of the problem domain knowledge, represented in an ontology, to define the user preferences. As we will discuss later, these preferences will take the form of new constraints that will be added to the problem to be optimized. In this way, the ontology plays a role as an interface to include new knowledge, defined in this case as problem constraints, into the optimization procedure of the multi-objective algorithm.

2.2. Ontology Concepts

In accordance with [7] and [8], an ontology provides a formal representation of the real-world. It defines a description of terms in a certain domain (classes or concepts), properties of each concept describing various features and attributes of the concept (properties), and restrictions on properties. That is to say, ontologies define data models in terms of classes, subclasses, properties, and subproperties. Using high-level formal representations like ontologies for declaring a given domain instead of a programming language provides some benefits [27]:

1. Ontologies are closer to natural language, so, it is easier for humans to express their thoughts.
2. Ontologies provide a more coherent and easy navigation from one concept to another and these concepts are like humans perceive the world.
3. Ontologies allow the automated reasoning about their data.
4. Ontologies are designed to be extended because relationships and concepts can be easily mapped to others in existing ontologies.

Ontologies are used for many different purposes, and they can be constructed and structured in many different ways [28]. One of the most common ways to describe the generalisation level of an ontology is by using the structure [8]. The structure depicts how a general domain ontology can be specialised into a domain ontology or a task ontology. As has been indicated above the domain ontology focuses on concepts which belong to a specific field like smart-cities. However, top-level ontologies represent general terms about a specific field, that is to say, domain ontology indicates, for example in the smart-city domain, elements like street, building, etc. Nevertheless, top-level ontology includes elements like spatial object (for defining an object that is located in a location) that is common across different domains.

The Ontology Web Language (OWL) defines ontologies in the context of the Semantic Web, and was approved by the World Wide Web Consortium (W3C) in 2004. It is based on RDF (Resource Description Framework) [29] and semantically extends RDF Schema.

From a formal description, OWL is equivalent to a very expressive description logic (DL)[30]. OWL is built on top of RDF data; in other words, the OWL specification defines exactly what can be written with RDF to make it possible to have valid ontologies. OWL ontologies are often published online and may refer to or be referred from other OWL ontologies.

2.3. The BIGOWL Ontology

BIGOWL [14] is an ontology for describing the components that could take part of Big Data analysis processes, including machine learning and optimization algorithms, workflow definitions, and the data used in the analysis. It is oriented not only to the semantic annotation of Big Data sources, components, and algorithms, but also to include domain knowledge to be used in the applications. In this work, we make use of classes and properties defined in BIGOWL for multi-objective optimization with metaheuristics. Figure 2 depicts this subset of BIGOWL regarding optimization algorithms. This ontology contains the declaration of well-known optimization techniques such as exact algorithms, heuristics and metaheuristics. More specifically, it focuses on describing metaheuristics families like *Population Based Methods* (Search, Swarm Intelligence and Evolutionary methods) or *Trajectory Methods*. Furthermore, BIGOWL also describes optimization problems such as: TSP or RND.

2.4. Smart city ontologies

The main concepts regarding to urban domain (route, buildings, location, position, communication, antenna, neighborhood, traffic, etc.) have been defined in several proposals for smart city ontologies [31, 32] and standards such as: Smart Cities-European Medium-Sized Cities[2], Open and Agile Smart Cities[3] or Smart Cities Council[4].

This heterogeneity has opened up the opportunity to create common models to allow a standard description of cities [33]. In ontological engineering, these models are known as Ontology Design Patterns (ODP). ODPs are inspired by software design patterns and can be understood as a reusable modeling solution to a recurrent ontology design problem [34]. In the standards, there are concepts like Administrative Area or City Object that are focused on describing how a city is organized (road, district, neighborhoods, etc.) and they describe the elements that can be found in a city: building, tunnel, bridge, etc. As a consequence, those patterns provide us with the classes required to describe an urban domain, reusing existing knowledge in this context.

3. Domain Ontology: Urban

For the design of the domain ontology, a series of competency questions were considered to follow the best practices and principles in ontological engineering [35]. Without loss of generality, these questions are formulated in the context of the urban domain of knowledge worked as a proof of concept, although it could be conducted on other different domains (live sciences, logistics, industry, etc.). The questions are:

- What is the set of characterizing classes for urban domain?
- What is the set of object properties for urban domain?
- What is the set of data properties for urban domain?
- Are there ontologies which Urban ontology can be aligned with?
- How do we align smart city and optimization domains with urban domain?
- How do we inject the domain knowledge in the process of optimizing multi-objective problems?

As a result, the Urban ontology [5] proposed here as domain is based on the ODPs defined in [33], in concrete *Administrative Area* and *City Object* ODPs. In order to reuse existing knowledge, we have aligned our ontology with a set of other existing ones specified in Table 1, which were also pointed out in Section 2.4. In addition, the Urban ontology re-uses the concepts related to optimization (algorithms, problems, etc.) defined in BIGOWL ontology. Figure 3 depicts the domain ontology organization and how it is related with other ones, including the design pattern of Administrative Area and City Object. The Urban ontology describes classes and properties for the two use cases in this work: TSP and RND; although it can be updated with additional classes to define other new use cases.

Figure 4 depicts the classes and properties defined in Administrative Area and City Object patterns for describing a city and its elements. The main classes of these patterns are:

- ***SpatialObject***. This class has been added in the patterns to define a super class that encompasses any geospatial[6]

[2] `http://smart-cities.eu`
[3] `http://www.oascities.org`
[4] `http://smartcitiescouncil.com`)

[5] Urban ontology:`https://github.com/KhaosResearch/urban/urban.owl`
[6] `http://www.opengis.net/ont/geosparql#`

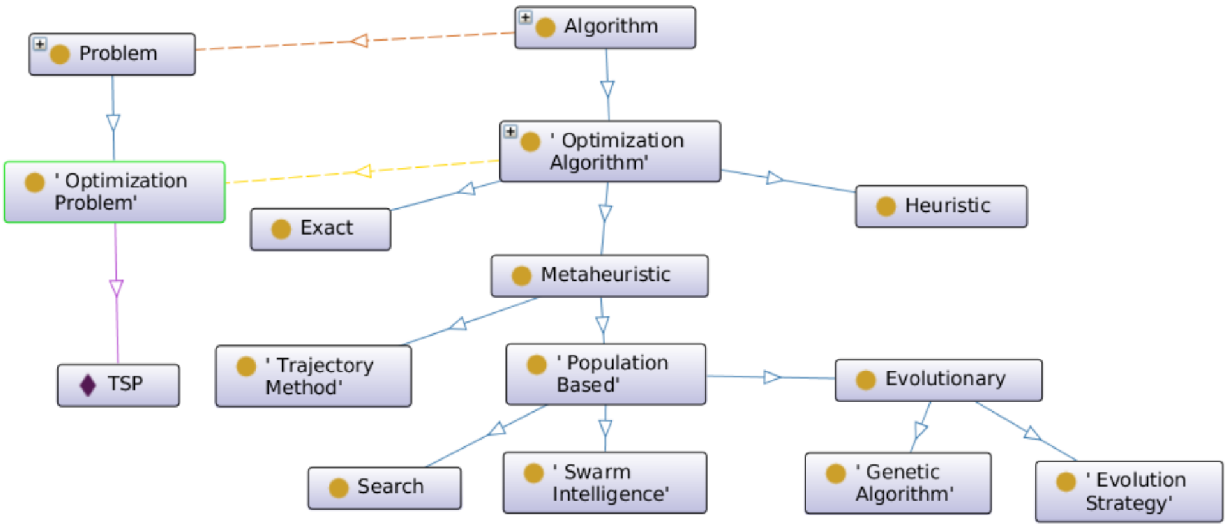

Figure 2: Representations in BIGOWL of optimization problems and algorithms

Table 1: Urban ontology is aligned with the following smart city ontologies. The column Class/Property indicates the terms defined in Urban Ontology.

Name	Reference	URI	Class/Property
km4city, the DISITKnowledge Model for City and Mobility	[20]	http://www.disit.org/km4city/schema#	hasGeometry
geosparql, Geometry spatial Model	[21]	http://www.opengis.net/ont/geosparql#	Geometry, Point, City, CityObject, contains
sao,Stream Annotation Ontology	[36]	http://iot.ee.surrey.ac.uk/citypulse/ontologies /sao/saov06.rdf	hasLocation
gci, Global City Indicator Foundation Ontology	[37]	http://ontology.eil.utoronto.ca/GCI/Foundation/GCI-Foundation.owl	Administrative Area

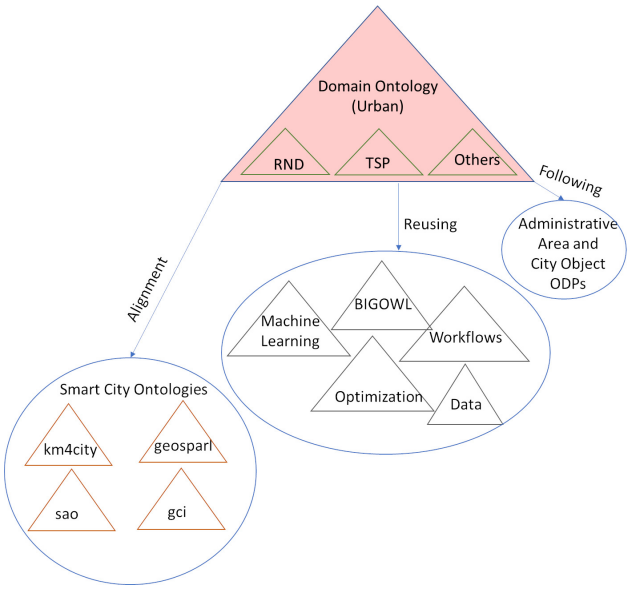

Figure 3: Urban ontology re-uses classes and properties from BIGOWL and is aligned with smart city ontologies. Furthermore, Urban ontology follows the Ontology Design Patterns of Administrative Area and City Object.

element.

- ***SpatialFeature***. It has been defined as a spatial class that represents anything with spatial extent. This class is described in the W3C standard *Spatial Data on the Web Best Practices*[7].

- ***Geometry***. The Geometry concept is included in geosparql ontology and has been used to represent the geometry, which defines a SpatialFeature (e.g., point, line, polygon, etc.).

- ***Point***. This class is also taken from geosparql ontology and represents where a certain element is located.

- ***AdministrativeArea***. This class represents places delineated for jurisdiction purposes of a particular government such as city, district, or neighborhood. AdministrativeArea class is based on the *Territory Vocabulary*[8] and the AENOR norm UNE 178301:2015[9]. AdministrativeArea class and its sub-classes are defined in different

[7] `https://www.w3.org/TR/sdw-bp/#dfn-spatial-thing`

[8] Territory Vocabulary: `http://vocab.linkeddata.es/datosabiertos/def/sector-publico/territorio`

[9] Norm UNE: `https://www.aenor.com/normas-y-libros/buscador-de-normas/une?c=N0054318`

ontologies such as, gci[10], km4c[11] and smart-city[12].

- ***City***. It comes from geosparql ontology and represents a City that can be defined by a geometry. A city can have city objects (e.g. buildings, bridge, etc.).

- ***CityObject***. The class CityObject represents the same class defined in the CityGML data model[13], which is an open standardized data model for digital 3D models of cities and landscapes. In the CityGML model, several modules are defined in order to represent the class definitions for the most important types of objects within 3D city models. Some sub-classes from CityObject are defined in the geosparql ontology as well.

Administrative Area and City Object patterns also define some properties for connecting their classes, as we can see in Figure 4:

- ***hasGeometry***. Spatial Feature elements are defined by a geometry. This property has been obtained from km4city ontology.

- ***hasLocation***. Spatial Feature elements can be located at a given point. The property comes from sao ontology.

- ***contains***. This property comes from geosparql ontology and its aim is to indicate that a city has different objects such as, buildings, tunnels, etc.

Taking all these into account, we have followed the standard Ontology 101 development process [35] to develop our ontology, in a inter-disciplinary approach between ontology and optimization experts. This process comprises seven steps:

1. *Determine the domain and scope of the ontology*. The scope is to define an ontology that describes, in this case, urban domain. It will allow us to annotate any problem related to that domain. In particular, it focuses on traveling salesman problem and antennae location design, which are defined in the context of a city and are solved using optimization algorithms.
2. *Consider reusing existing ontologies*. The ontology reuses classes and properties from BIGOWL and is aligned with a set of smart city ontologies. On one hand, BIGOWL has been validated to describe the concepts related to optimization (algorithms, problems and constraints). On the other hand, the set of smart city ontologies defines, among other concepts, key terms in a city such as, city objects (building, antenna, tunnel, etc), location, route, etc.
3. *Enumerate important terms in the ontology*. Important terms were selected from the related literature, such as: *Optimization problem*, *Route*, *Preference*, *Position* in the path, *Buildings* and their *Location* where the *Antenna* will be installed.
4. *Define the classes and the class hierarchy*. Having the BIGOWL and smart city ontologies, the main terms, listed above, are defined in our ontology.
5. *Define the properties of classes and slots*. With the purpose of definning relationships between classes and defining attributes, we have made use of the BIGOWL data properties. Nevertheless, we have defined new data properties to indicate, for example, buildings location through X, Y coordinates.
6. *Define the facets of the slots*. As this step includes the definition of cardinality constraints and value restrictions for the ontology's properties, we have defined one to indicate that the route position only can be an integer.
7. *Create instances*. In order to describe the domain of the problems, the ontology contains some instances such as city, route, user preference nodes, sub-route, antennae, buildings, location, etc.

Figure 5 shows the main classes of our domain ontology (Urban). Wherever possible, its classes are aligned with those of the smart city ontologies (Table 1). Classes in grey background are related to optimization, so they come from BIGOWL, while classes denoted with dotted line squares describe spatial and geo-spatial terms, thus they are aligned with geosparql ontology. Some object properties are included from km4city ontology (hasGeometry) and sao ontology (hasLocation). Finally, solid line classes are defined in our Urban ontology and are its core classes.

In addition to the classes described in the Administrative Area and City Object patterns, Urban ontology includes new classes such as:

- ***Grid***. This class represents the spaced horizontal and vertical lines used to identify locations on a map.

- ***GridConstraints***. With this class the analyst or decision maker specifies whether a location in the grid is allowed to be occupied or not.

- ***Antenna***. It specifies a new object to consider communication antennae in a city.

- ***Route***. This class represents a route in a city. A route is constituted by city objects, usually building instances.

- ***Preference***. With this class, the DM or the analyst indicates their preferences for instance, the order of visiting a node in a route, sub-routes, etc.

- ***Position***. It represents the order of a node in a route. Usually a position can be specified in user's preferences.

[10] gci:`http://ontology.eil.utoronto.ca/GCI/Foundation/GCI-Foundation.owl`

[11] km4c:`http://www.disit.org/km4city/schema#`

[12] smart-city: `https://www.dropbox.com/s/q7tz39jjeeibhzl/2015-SMART%20CITY%20ONTOLOGY-V01.owl?dl=0`

[13] CityGML: (`https://www.citygml.org/about`)

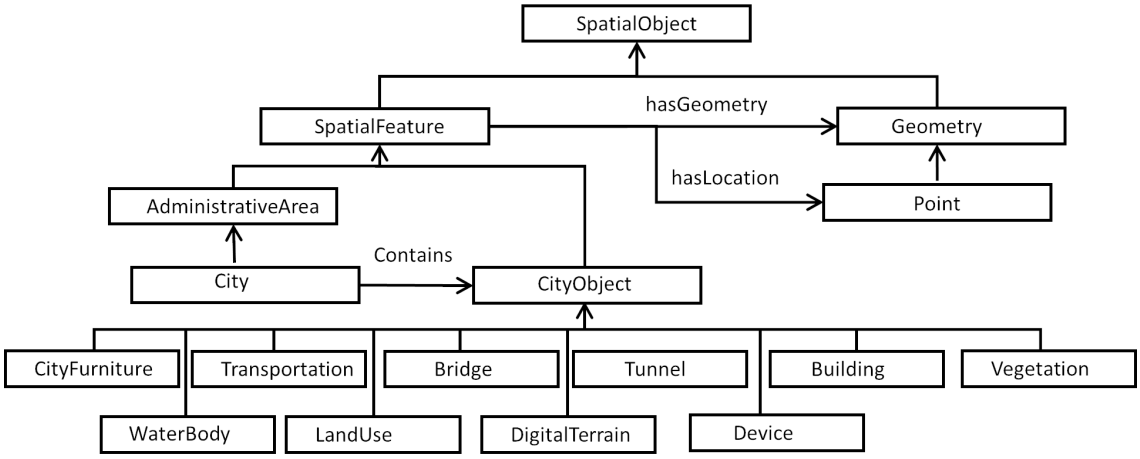

Figure 4: Administrative Area and City Object patterns provide a model for an object that belongs to a city.

4. Injecting Domain Knowledge Into Code

Once the domain ontology is specified and the DM has defined his/her requirements, the next step is to set them within the optimization process. The idea is to properly update constraints or preferences in the problem model, so as to later take advantage of them in the optimization process. We have selected the jMetal Java-based multi-objective optimization framework as the optimization engine [18][19].

In this framework, all the problems have to inherit from interface *Problem*. This interface contains the basic methods related to getter methods to retrieve the main properties of a problem (i.e, number of decision variables, number of objective functions, number of side-constraints, and problem name), a method for creating a solution, and a *evaluate()* method which, given a solution, evaluates it. The code snippet of this interface is:

```
public interface Problem<S> {
  int getNumberOfVariables() ;
  int getNumberOfObjectives() ;
  int getNumberOfConstraints() ;
  String getName() ;

  void evaluate(S solution) ;
  S createSolution() ;
}
```

As we want to expand the methods of the problem, we have created a new interface extending *Problem* called *DynamicConstraintProblem*, which includes an *addConstraint()* method aimed at allowing to add constraints dynamically, after the problem has been instantiated (see the code snipped below):

```
public interface DynamicConstraintProblem<S> extends
    Problem<S> {
  addConstraint(Function<S, Double> constraint);
}
```

The strategy adopted basically consists in automatically mapping every user's requirements into new constraints, which are attached to the problem to optimize. jMetal includes a constraint handling mechanism based on penalizing those solutions that are unfeasible because some constraint is violated. As the user preferences can vary depending on the criteria of the domain expert, we want to avoid having to modify the problem code each time a new constraint has to be added.

The procedure for injecting domain knowledge into the problems is a collaborative work between ontology and optimization experts, comprising the following steps:

1. The optimization expert implements the problem in jMetal, which must extend the template. This implies that the encoding of the solutions have to be defined.
2. The ontology expert defines the user's requirements in OWL by using classes, object properties, and data properties described in the domain ontology (both the BIGOWL and Urban ontologies in our use cases). The result of this step is a set of user preferences templates that can be shared and instantiated by different final users to specify its own preferences and constraints.
3. For each user's requirement template defined in the previous step, the optimization expert writes the corresponding code which mapping it to a constraint in jMetal. All the user's requirement templates use the same classes and properties and consequently, the same code is able to process several instantiated users' requirements. This is developed through functions, which given a solution, checks whether it is feasible or not. In the latter case, it returns a value representing the violation degree. All the user requirements are processed with the OWLAPI library [38] and their associated constraint code is automatically generated, leading to a list of constraints.
4. Once the problem is instantiated by the DM, the list of constraints can be added "on the fly" to the problem.
5. The optimization algorithm is executed, taking into account the user constraints whenever a solution has to be evaluated.

To illustrate the full processes, let us consider a generic example, where the user is interested in establishing a preference

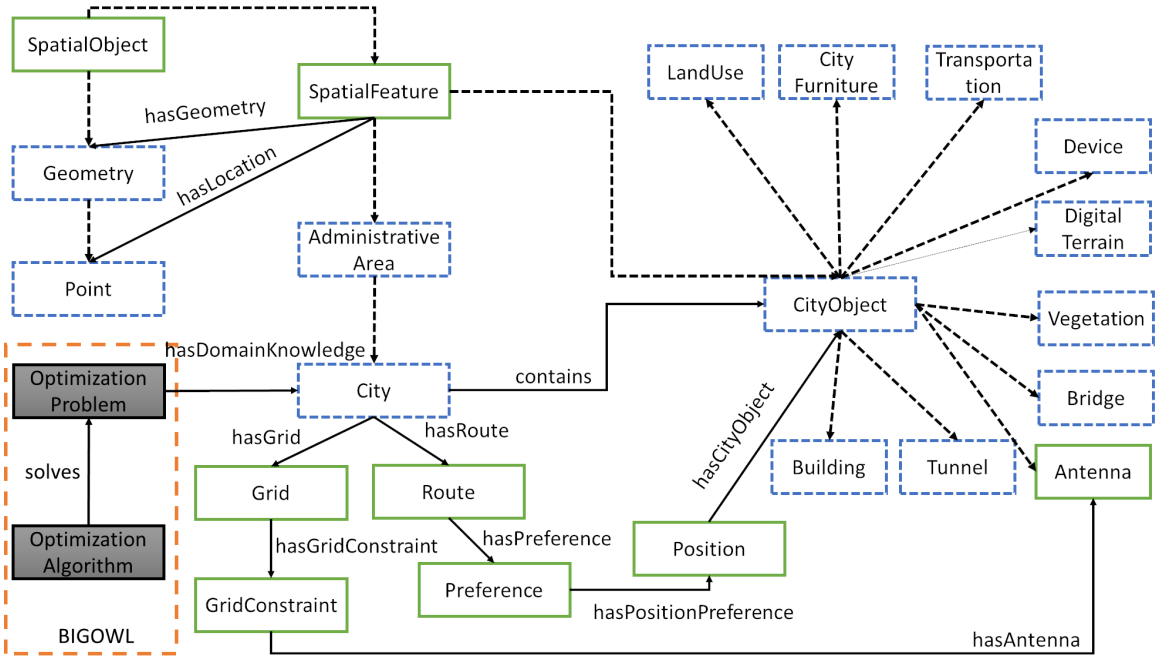

Figure 5: Overview of the classes of the Urban ontology and BIGOWL ontology. Terms in solid line are defined in the urban ontology, dotted line ones are aligned with smart city ontologies (geosparql and gci). Dotted arrows refer to sub-classes and normal arrows refer to object properties. Furthermore, terms in gray ones are defined in BIGOWL ontology.

where the position of two particular elements in a permutation should be in a row, that is to say, one after the other in a given order. According to this, Step 1 is to implement the class of the problem. We consider that each solution is a permutation of element positions.

Step 2 consists in defining this preference in RDF with regards to the Urban ontology. In this step, the preference is defined as a template: an individual of the class *Preference* is created, then its properties can be used to specify the user preference. Table 2 shows the corresponding RDF code, where two permutations elements *ElementA* and *ElementB* are created as individuals of the class *Bulding* and the properties of the classes *Point* and *Position* are used to assert their position and location in the map. Then, the desired order of the two elements is established with the properties *isFirst* and *isNext*. After this, the template can be instantiated to specify a concrete order of different elements in different positions without changing any of the classes and properties. Therefore, only one mapping function is needed to generated the constraint in jMetal. It is worthy to highlight how the ontology provides flexibility, generality and abstractness when describing user preferences.

Step 3 comprises the implementation of the code that generates a function to evaluate the constraint representing a specific preference. Each constraint has a method called *createConstraint()*, which returns a function that, given a solution, checks whether it is feasible or not. In the latter case, it returns a value representing the violation degree. Each solution is encoded as a permutation including the element positions, which will be located in the order they appear in the permutation. An example of the code to implement is snippet below, assuming that the two elements are referred to as A and B:

```
1  Function<Solution, Double> createConstraint(int elementA,
       int elementB) {
2
3    Function<Solution, Double> constraint = solution -> {
4
5      int indexA =
           solution.getVariables().indexOf(elementA) ;
6      int indexB =
           solution.getVariables().indexOf(elementB) ;
7
8      if (indexA == (indexB - 1))
9        return 0.0;
10     else
11       return (-1.0 * Math.abs(elementA - elementB))
12   }
13
14   return constraint;
15 }
```

We can observe that the *createConstraint()* method has two parameters, *elementA* and *elementB*, representing the elements that should be consecutive in the permutation. The method defines a function (lines 3-13) that, given a solution, returns a double value indicating its violation degree. A permutation in jMetal is encoded in a solution having as decision variable consisting of a list of integers, so the function finds first the index of the two nodes in the permutation (lines 5 and 6). Then, it checks whether the index of node A is just after the index of node B (line 8); if so, a value of 0.0 is returned (i.e., the solution does not violate the constraint), else the result is a penalty computed as the distance of both positions multiplied by -1 (because the violation degree of an inequality constraint must be a

Table 2: Instances of variable position examples in OWL Manchester syntax.

Terms	OWL Manchester syntax
InARowPreference	Individual: InARowPreference Types: urban:Preference Facts: urban:hasPositionInPreference ElementAPosition, urban:hasPositionInPreference ElementBPosition
ElementAPosition	Individual: ElementAPosition Types: urban:Position Facts: urban:hasCityObject ElementA, urban:hasNext ElementBPosition, urban:isFirst true
ElementA	Individual: ElementA Types: geosparql:Building Facts: sao:hasLocation ElementALocation, urban:hasId "86"^^xsd:int
ElementALocation	Individual: ElementALocation Types: geosparql:Point Facts: urban:hasX "65"^^xsd:int, urban:hasY "22"^^xsd:int
ElementBPosition	Individual: ElementBPosition Types: urban:Position Facts: urban:hasCityObject ElementB
ElementB	Individual: ElementB Types: geosparql:Building Facts: urban:hasId "93"^^xsd:int
ElementBLocation	Individual: ElementBLocation Types: geosparql:Point Facts: urban:hasX "65"^^xsd:int, urban:hasY "25"^^xsd:int

negative value, according to the formulation of multi-objective problem defined in Equation 1). This function for creating constraints give us the possibility of including several constraints, as all the users' preferences are defined following the same template. It is also interesting to note that if the users' preferences are not described through the semantic annotation, all the preferences have to be implemented from scratch.

```
1  PermutationProblem<> problem = new MultiobjectiveTSP(...);
2
3  owlUtils.loadOntology("ontology.owl");
4  List<Function<Solution, Double>> constraintList =
5        owlUtils.getConstraintsFromOntology();
6  for(Function<Solution, Double> constraint: constraintList){
7    problem.addConstraint(constraint);
8  }
9  algorithm = new NSGAII(problem, ...) ;
10 algorithm.run() ;
11 List<Solution> resultPopulation = algorithm.getResult();
```

Finally, according to the code snippet next, once the problem is created (line 1), Step 4 consists in loading the ontology (line 3) and creating a list of constraints functions from the preferences defined in the ontology (lines 4, 5). The incorporation of these constraints to the problem is iterated in a list (lines 6-8). Finally, the last step is to create an algorithm that is executed to solve the resulting problem, producing as a result a list of solutions representing the Pareto front approximation found (lines 10-13).

5. Use cases

As commented before, to validate our proposal we have elaborated two use cases, based on bi-objective formulations of the Traveling Salesman Problem (TSP) and the Radio Network Design (RND). Both are assumed to be defined in an urban context. For each optimization problem, we describe how its main elements are annotated in the domain ontology (Urban), including encoding, preferences, constraints, experiments and results.

5.1. Use Case 1: Traveling Salesman Problem

The well-known TSP consists in finding a single route in a graph, starting and ending in a same node, covering all the nodes which must be visited only once. The formulation used defines two conflicting objectives: minimizing the travel time and minimizing the total traveled distance. As commented in the previous section, a TSP solution is encoded as a permutation of nodes, so the user preferences about routes must be defined having this representation in mind.

As the context of the TSP is an urban environment, we assume that a traveler has to deliver some kind of goods to clients who are in a city following a delivery route. Each client is in a building that is located in a position. In this use case, the user is interested in defining five preferences, leading to individuals in the domain ontology shown in Figure 6.

The five user preferences are grouped in two categories: *fixed position preferences* and *variable position preference*. All the

Table 3: TSP instances of fixed position examples in OWL Manchester syntax (VIP preference example)

Terms	OWL Manchester syntax
VIPPreference	Individual: VIPPreference Types: urban:Preference Facts: urban:hasPositionInPreference VIPPosition
VIPPosition	Individual: VIPPosition Types: urban:Position Facts: urban:hasCityObject VIP, bigowl:hasDataType "integer"^^xsd:string, urban:hasPositionValue "1"^^xsd:int,
VIP	Individual: VIP Types: geosparql:Building Facts: sao:hasLocation VIPLocation, urban:hasId "78"^^xsd:int
VIPLocation	Individual: VIPLocation Types: geosparql:Point Facts: urban:hasX "10"^^xsd:int, urban:hasY "20"^^xsd:int

instances are defined as instances of *Preference class*. We describe next their properties and their semantic meaning. Tables 8 and 9 in Appendix depict the Manchester syntax of all the instances of this use case.

5.1.1. Fixed Position Preferences

A kind of preferences can be related to particular nodes (buildings) which are are located at fixed positions in a route. Three preferences of this type are defined: a node must be in the first position of any route, another one must be in the middle, and a third one must be in the last position.

We highlight now the classes and instances used for describing these examples, included in Figure 6. *VIPPreference* indicates when a VIP client must be visited in a particular position in the route. *VIPPosition* is an instance of *Position* class, which defines that position in the route. A *VIP* instance is created as a *Building* class, that is to say, each node of the route represents a building. *VIPLocation* is aimed at indicating the position of VIP client in the city, and it is an instance of class *Point*. The other two preferences are *HalfWayClientPreference* and *LastClientPreference*. Along with *VIPPreference*, they include *hasPositionValue* to indicate a position in the route. *hasPositionValue* can contain a number or a percentage, so the Urban ontology has the data property *hasDataType*, which indicates the data type (an integer in the three considered examples).

Table 3 depicts the terms defined in Urban ontology to describe the examples of fixed positions preferences in a route related to the VIP preference. The terms of the other preferences are described in the Appendix in Table 8.

5.1.2. Variable Position Preferences

We consider here that the user may not be only interested in visiting nodes in fixed route positions, but in positions fulfilling some condition. For example, it would desirable that a node be visited among the 25% of the first ones in the route, or that if the traveler visits node A, which is at position x, then it must visit next node B in position $x + 1$. These two examples are described next:

- *FirstQuarterPreference* describes a preference where the node or client has to be visited in the first quarter of a route. Here *FirstQuarterPosition* indicates a percentage, and the position of the client is related to the length (*hasLength*) in the route.
- *SubroutePreference* simulates the case where the deliverer, after visiting a bakery, must go to a given restaurant. Thus, the restaurant position must be the *next* of the bakery position. In this case, the ontology does not use *hasPositionValue* data property, because the number of the position in the route is not important, as the user only wants that client Bakery is visited just before the restaurant.

Table 4 depicts the terms defined in the domain ontology Urban, so as to describe the *SubroutePreference*. Table 9 in Appendix includes all the variable positions preferences.

5.1.3. Experimentation with the TSP

We conduct in this section an experiment to analyze whether the preferences set in terms of variable positions conduct the expected results. Concretely, we are interested in observing how the presence of preferences alters the Pareto front approximations and in verifying that the resulting routes are feasible solutions according to the constraints previously defined.

To this end, the algorithm used is the well-known multi-objective evolutionary algorithm NSGA-II [39] with the following parameter settings: the population size is 100 and the variation operators are partially-mapped crossover (PMX) and swap mutation with application probabilities of 0.7 and 0.02, respectively. The stopping condition is to compute 1,000,000 function evaluations. We have defined two 100-city bi-objective instances of the TSP, taking values of distance and time travel from academic benchmark problem instances kroa100 and krob100 for the instance 1 and kroa100 and kroc100 for the instance 2 from the TSPLIB [40].

The preferences are annotated in the ontology, so the model is able to induct them in the specific problem automatically. These concrete preferences are as follows:

1. VIP node: 78.
2. Node 51 must be in the middle of the route.
3. Node 89 must be the last one.
4. Node 34 is in the first quarter part of the route (between positions 1 and 25).
5. Subroute: nodes 52 and 9 must be consecutive.

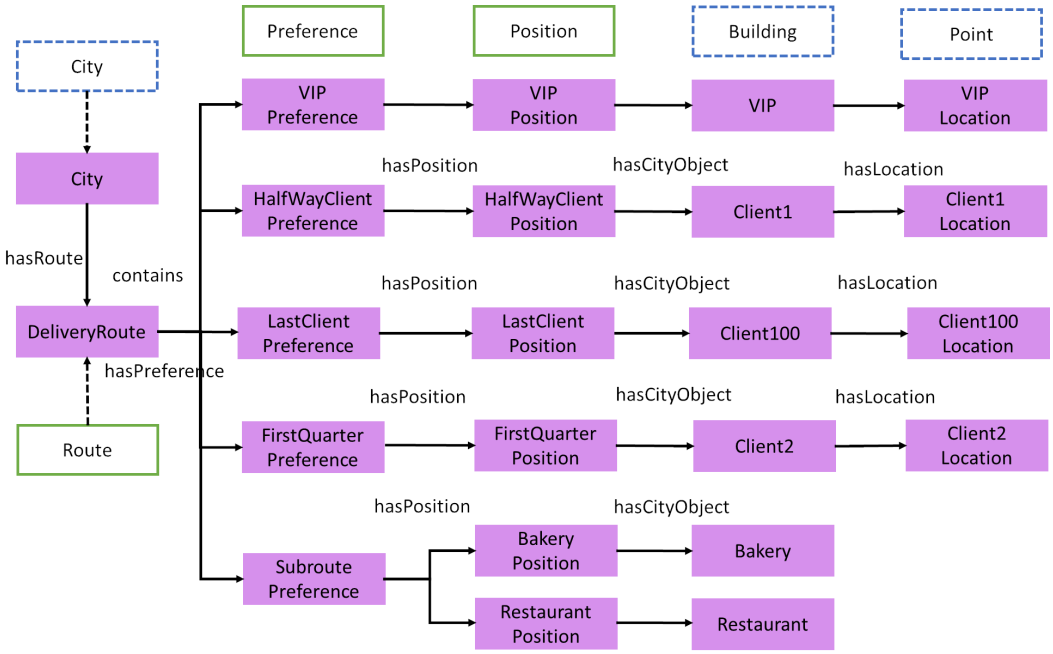

Figure 6: Overview of the individuals in the domain ontology "Urban" of the TSP problem. Dotted arrows refer to sub-classes and normal arrows refer to object properties.

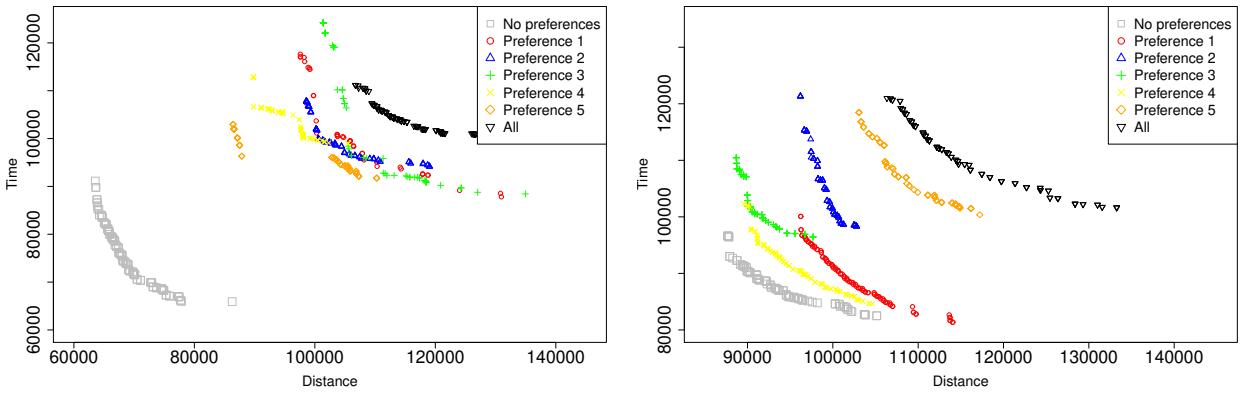

Figure 7: Pareto front approximations obtained when solving bi-objective TSP problem without preferences (constraints), with each of the five defined ones, and with all of them. On the left instance 1 and, on the right instance 2.

Figure 7 shows the Pareto front approximations produced by NSGA-II when solving the two TSP instances each one with seven combinations of preferences (generated as different versions of the Urban ontology, without needing to change the software implementation of the problem): no preferences, each of the five defined preferences individually, and all the preferences at the same time. We can observe that, as expected, the front of the problem without preferences dominates the rest of fronts, as adding preferences penalizes the distance and time objectives. The front ranking the second is the one corresponding to the fourth preference (the node must be in the range of positions between 1 and 25), which makes sense, as that preference is weaker than the other four. We also observe that the front corresponding to the combination of all the preferences is dominated by the rest of fronts, what is also a foreseeable result.

The second part of this experimentation is to validate that the preferences are taken into account in all the solutions included in the obtained Pareto front approximation. This is confirmed after checking that all the produced solutions are feasible for all the scenarios. For each scenario, we have selected the solution located in the middle of the front, and we include in Table 5 the route obtained for each configuration and instance, where we highlight the nodes of interest.

5.2. Use Case 2: Antennae Location Design

The Radio Network Design (RND) is an optimization problem found in telecommunications based on the deployment a set of antennae in a region to provide a service of mobile communications. Concretely, given the available positions for locating

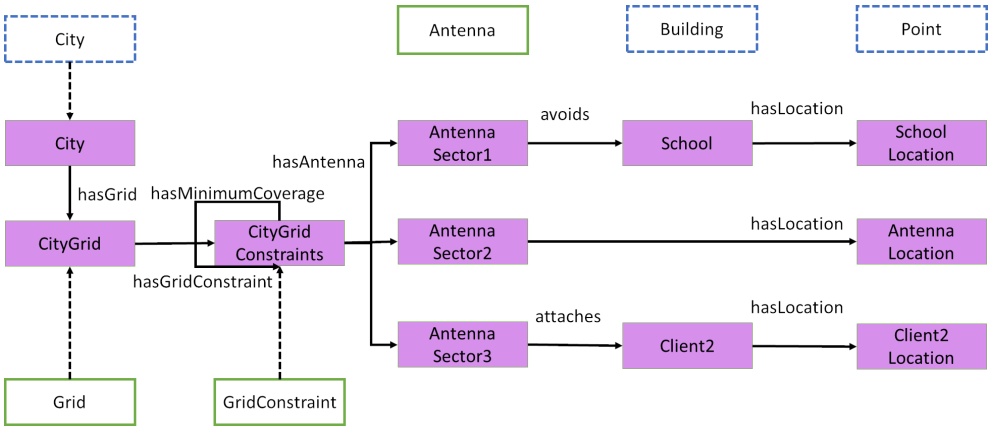

Figure 8: Overview of the individuals in the Urban ontology of the RND problem. Classes in dotted squares are from geosparql ontology, solid line classes are new classes in Urban ontology and instances have background color. Dotted arrows refer to sub-classes and normal arrows refer to object properties.

antennae in a particular area of a city, the problem consists in finding the set of locations having into account two conflicting objectives: minimize the number of antennae (to reduce cost) and maximizing the coverage (to give a satisfactory service). We use the RND formulation given in [41], where a binary string representing all the locations is used to encode the solutions (1 means an antenna is located there).

From the decision maker point of view, the preferences in the RND are related to fix or avoid antennae locations, which can be specified by selecting buildings or by indicating their coordinates. For example, the telecommunication engineer can decide to avoid placing antennae on schools or hospitals, and to deploy an antennae near a square to ensure and improve the service coverage around that place. In RND optimization problem, a decision maker is able to define preferences related to the total coverage in the grid.

Figure 8 shows the terms that describe this use case. The classes from the geosparql ontology are in dotted square, the new classes that have been created in Urban ontology are in solid square, and their instances have background color. The main classes are Antenna (that represents an antenna in the grid), Building (which defines the place of an antenna, e.g. client building or school, etc.) or Point (that indicates the coordinates of a Antenna or a Building). In Figure 8, different antennae locations are defined, e. g., *Client2* represents the position of a client's building in the grid.

5.2.1. Antennae positions based on building's locations

We assume here that the user may be interested in avoiding or placing antennae based on the positions of buildings, such as: high schools, parks, hospitals, etc. For example, it must avoid placing an antenna near to a building. However, another client may desire an antenna to be set at that building. These two examples are described next:

- *Avoiding school* simulates the case where a user disallows the position of one or more buildings in the grid for placing antennae.
- *Placing antenna in a building* describes the case when the antenna position is fixed by the position of a building. Like in the other example, we can use more than one building or fixing more than one antenna.

5.2.2. Antennae positions based on their coordinates

The antennae positions can be defined by explicitly indicating their coordinates, that is, the user can indicate to place an antenna in a grid position where there no exists any *city object* such as: building, tunnel, etc. However, it would be possible that the user does not know the city objects and only knows their coordinates.

Table 6 contains the terms in OWL Manchester syntax for RND use case when placing an antenna near to a school is avoided . City grid definitions and all the examples of antenna preferences are organized in Table 10. As we can observe in this table, the example related to *Fixed Antenna Location by Client Location (Antenna Sector 3)* the instance Client2 also appears in the sales agent routing use case. So, the domain ontology Urban is able to describe more than one use case and can re-use its instances in multiple use cases.

5.2.3. Experimentation with the RND

For this case, we use two RND instances, the first one is analyzed in [41], which is featured by 149 available locations and antennae providing a square coverage, and the second one is defined in [42] with 249 locations. As optimization algorithm, the NSGA-II is also used, although with different parameter settings in this case: the population size is 100 and the single point crossover operation is used with a probability of 1.0. In the case of the mutation operator, we choose the bit flip mutation with a probability of 0.01.

We assume the user has defined the following preferences:

1. Avoid to locate an antenna at the school: position 0.
2. Placing an antenna by its coordinates: position 10.
3. Placing an antenna at the position of client2: position 25.
4. The minimum coverage is the 50% of total coverage.

Figure 9 depicts the Pareto front approximations calculated using NSGA-II when solving the RND with no preferences (top

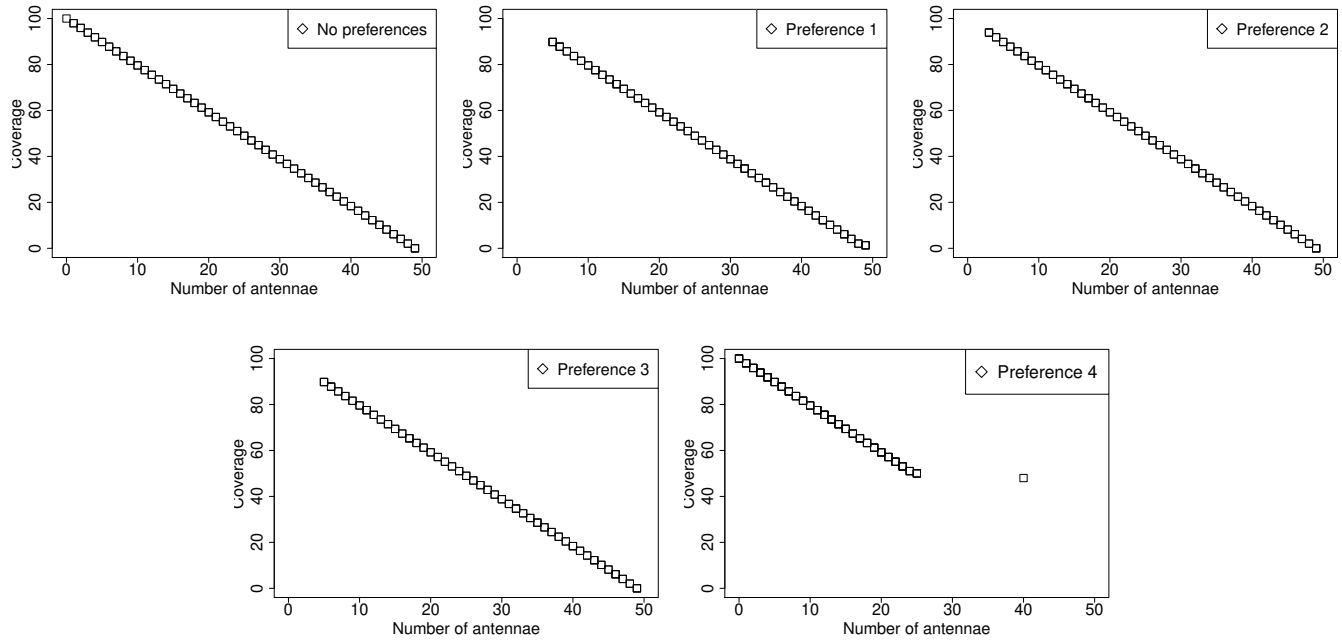

Figure 9: Pareto front approximations obtained when solving the first instance of the bi-objective RND problem without constraints (on the left top), preference 1 (on the right top), preference 2 (on the middle left), preference 3 (on the middle right) and preference 4 (on the bottom left).

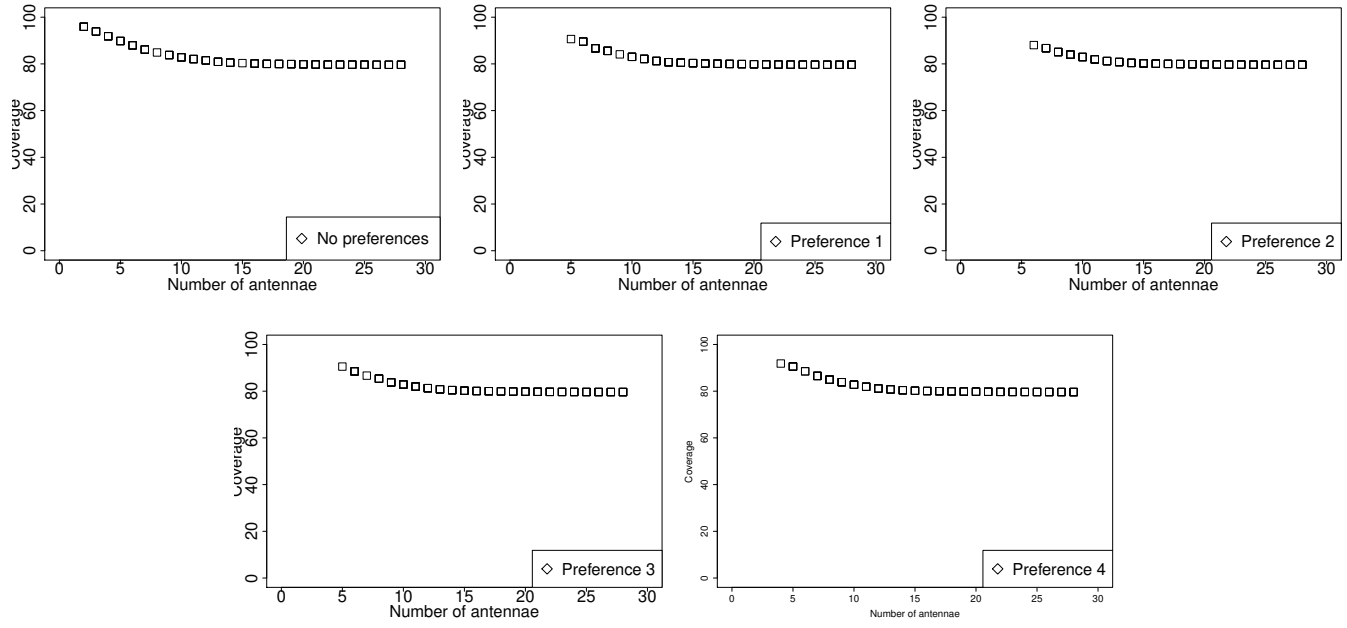

Figure 10: Pareto front approximations obtained when solving the second instance of the bi-objective RND problem without constraints (on the left top), preference 1 (on the right top), preference 2 (on the middle left), preference 3 (on the middle right) and preference 4 (on the bottom left).

left chart), as well as with each of the four preferences previously defined. As we can see when we are dealing with some preferences, the coverage of the Pareto fronts declined with respect to the problem without preferences. In this sense, we can easily check (sub-plot of preference 4) how the minimum coverage of 50% of the total coverage is induced in the Pareto front approximation. The fronts produced when solving the second instance are included in Figure 10; in this case, the differences in the fronts are more subtle that in the first case.

As in the case of TSP, we have checked that all the solutions obtained are feasible in all the scenarios. After selecting the solution in the middle of the front, Table 7 shows the corresponding solutions to the RND problem according to the defined preferences. The positions affected by these preferences

Table 4: TSP instances of variable position examples in OWL Manchester syntax (subroute preference example).

Terms	OWL Manchester syntax
SubroutePreference	Individual: SubroutePreference Types: urban:Preference Facts: urban:hasPositionInPreference BakeryPosition, urban:hasPositionInPreference RestaurantPosition
BakeryPosition	Individual: BakeryPosition Types: urban:Position Facts: urban:hasCityObject Bakery, urban:hasNext RestaurantPosition, urban:isFirst true
Bakery	Individual: Bakery Types: geosparql:Building Facts: sao:hasLocation BakeryLocation, urban:hasId "52"^^xsd:int
BakeryLocation	Individual: BakeryLocation Types: geosparql:Point Facts: urban:hasX "2"^^xsd:int, urban:hasY "2"^^xsd:int
RestaurantPosition	Individual: RestaurantPosition Types: urban:Position Facts: urban:hasCityObject Restaurant
Restaurant	Individual: Restaurant Types: geosparql:Building Facts: urban:hasId "9"^^xsd:int
RestaurantLocation	Individual: RestaurantLocation Types: geosparql:Point Facts: urban:hasX "3"^^xsd:int, urban:hasY "3"^^xsd:int

are underlined to point out the specific values (zero or one) that the semantic model sets for each case.

6. Discussion

User preferences and domain constraints can be hard-coded according to different criteria. The usual way is to include those preferences as problem constraints in their formulations. Another possibility is using hybrid techniques encompassed within the concepts of memetic algorithms, which implement operators designed specifically to take them into account [43]. In this work, we have used ontologies to support the automatic injection of user preferences and domain constraints in the optimization problem modeling, bridging the gap between semantic technologies and optimization algorithms. Specifically, we have carried out a first approximation covering the steps from the design of a simple domain ontology (including the domain knowledge of two use cases related to multi-objective problems) to the coding of an implementation in jMetal, obtaining Pareto front approximations with an evolutionary algorithm.

One of the main research findings we claim with the design and implementation of a domain ontology is the ability to represent and consolidate knowledge of a given optimization context. This knowledge-based approach allows us to annotate (i.e. to "semantize") all the meta-data involving data sources, user constraints and mathematical modeling. The meta-data are integrated following the ontology structure and stored in standard RDF triples, which are mapped with software code to implement such specific problem modeling and constraints. This methodology, extracts the problem modeling and constraint definition in such a way that changes in them only imply modifying the ontology instances (the ontology structure and the software code do not need to be updated). Thus, a given implementation can be easily reused in different scenarios. For example, the code for locating antennae in New York will be the same as for locating them in Madrid. The ontology instances have to be modified for each scenario, but the ontology structure remains the same.

In terms of practical implications, the proposed semantic model represents an initial demonstrator for the experimental piloting of the bi-objective TSP and RND problems with semantics. Nevertheless, thanks to the domain ontology abstraction, it could be easily adapted to other related problems (e.g., the Vehicle Routing Problem) and different problem domains, from graph modeling in logistics to biological networks.

7. Conclusions and Future works

In this paper, we have presented an ontology-based approach for the semantic annotation of problem domain knowledge and the practical injection or mapping of this knowledge into programming code. We combine the descriptive capacity of Semantic Web technologies with optimization problems and algorithms in order to facilitate the process of decision making by the human expert.

Table 5: Routes obtain obtained when solving bi-objective TSP problem using the user preferences

Case Study	Route instance 1	Route instance 2
VIP Client (node 78)	**78** 48 89 20 58 98 96 75 94 24 8 33 54 22 31 90 97 4 29 80 19 57 65 3 32 81 12 77 50 16 44 76 6 56 86 82 13 42 70 45 85 61 99 36 51 47 40 95 43 49 53 60 34 93 37 64 25 69 52 92 10 9 83 23 17 35 18 62 30 63 1 91 46 74 55 66 38 72 27 7 0 71 87 21 5 14 73 59 26 11 28 2 84 67 68 39 79 88 41 15	**78** 44 41 75 70 45 28 63 77 30 32 7 58 3 18 76 89 0 73 15 87 42 51 13 34 2 94 4 99 84 52 68 20 85 67 61 25 91 22 54 49 47 95 40 82 14 71 29 79 39 72 38 5 50 62 17 16 43 35 23 31 60 11 6 57 97 37 65 64 93 98 90 10 27 66 19 74 21 92 96 69 55 1 81 12 26 24 9 53 46 80 8 56 36 83 59 86 88 33 48
Client in middle of the route (node 51)	0 92 84 72 27 10 90 96 7 30 62 23 22 59 42 6 56 70 94 2 28 75 60 57 88 41 17 87 21 93 98 9 52 69 66 61 55 99 68 4 38 39 67 80 95 40 77 47 24 36 **51** 32 81 12 1 13 49 63 50 82 86 33 8 45 26 11 73 58 31 97 20 79 29 19 89 34 14 5 3 65 53 54 76 43 91 18 15 83 71 64 25 74 37 35 16 44 46 78 48 85	43 79 7 67 6 48 0 5 3 14 81 77 12 95 40 44 76 49 45 8 56 36 84 80 66 98 70 33 99 2 72 27 58 94 75 29 47 32 54 28 38 4 13 25 59 71 37 89 96 55 74 **51** 10 31 52 60 26 24 39 68 42 61 97 90 85 11 19 46 78 93 22 53 82 16 91 15 83 9 92 50 41 30 88 63 86 34 73 64 18 21 65 69 20 87 17 23 35 62 57 1
Client in the last position of the route (node 89)	90 8 6 11 2 45 44 76 59 73 93 96 98 18 91 30 66 92 0 79 41 5 14 31 64 74 27 72 68 24 9 52 78 7 88 63 50 40 70 49 48 71 58 16 56 42 82 15 87 46 19 85 26 22 34 95 53 39 69 25 55 61 80 60 86 57 1 81 12 43 35 17 23 21 4 94 36 29 99 54 51 32 13 47 77 33 28 75 67 38 84 10 97 20 83 65 3 62 37 **89**	9 41 88 39 72 24 2 8 2 12 80 4 49 0 92 90 51 94 75 32 26 8 85 58 10 96 27 7 37 36 43 1 5 98 23 18 50 81 63 60 57 53 68 61 66 46 64 87 22 16 71 83 48 78 93 21 17 35 15 25 79 73 31 11 67 29 19 76 44 14 3 65 91 62 86 54 40 42 45 99 38 84 20 52 69 97 34 56 70 47 95 77 59 13 82 33 6 30 55 74 **89**
Client in the first quarter (node 34)	68 39 38 61 8 24 45 29 99 4 94 75 28 11 2 92 84 72 67 70 42 13 82 **34** 26 65 3 5 88 57 83 73 93 37 48 9 20 98 58 31 90 97 96 7 41 30 81 1 44 14 59 60 32 47 40 91 21 22 54 51 53 36 27 10 89 69 25 55 74 71 6 33 86 50 62 76 23 15 87 78 79 64 46 80 19 85 0 56 49 63 43 95 77 12 16 35 17 18 52 66	67 49 21 78 89 65 88 93 37 26 56 59 **34** 17 16 23 87 15 9 20 73 82 70 47 4 36 33 13 99 12 50 80 53 68 96 28 75 0 46 3 35 98 58 95 40 77 24 57 86 42 63 41 44 83 48 7 18 74 14 31 92 60 32 81 51 72 54 76 43 1 5 62 30 55 66 84 52 69 22 85 19 79 61 97 90 27 6 91 11 39 94 8 25 64 71 10 45 2 38 29
Sub-route (nodes 52 and 9)	94 36 51 32 81 14 73 5 44 40 47 50 30 41 63 53 68 39 67 38 61 19 8 56 6 54 86 82 49 43 1 76 22 46 78 71 83 88 79 74 48 24 57 91 18 17 23 35 16 62 12 77 95 13 42 70 2 28 75 84 72 0 89 20 93 65 3 64 25 69 96 7 98 58 31 10 **52 9** 37 15 21 87 55 66 92 27 90 97 11 34 59 60 26 33 45 85 80 29 99 4	69 41 88 39 72 24 28 2 12 80 4 49 0 92 90 51 94 75 32 26 8 85 58 10 96 27 7 37 36 43 1 5 98 23 18 50 81 63 60 57 53 68 61 66 46 64 87 22 16 71 83 48 78 93 21 17 35 15 25 79 73 31 11 67 29 19 76 44 14 3 65 91 62 86 54 40 42 45 99 38 84 20 **52 9** 97 34 56 70 47 95 77 59 13 82 33 6 30 55 74 89
All preferences	**78** 74 53 49 1 63 62 83 23 37 80 29 77 12 **34** 81 7 96 93 65 3 5 14 73 59 32 86 50 82 42 13 40 47 95 43 91 18 35 16 44 17 **52 9** 87 31 97 61 19 85 2 **51** 70 45 26 28 36 75 94 38 4 99 54 76 22 46 0 48 71 20 98 58 10 90 27 92 57 24 60 33 6 56 8 11 84 67 72 68 39 66 55 64 15 41 30 88 79 25 69 21 **89**	**78** 98 56 8 64 6 57 74 48 62 91 88 82 42 54 40 47 77 49 33 45 2 **34** 38 66 15 65 30 41 39 61 46 36 80 12 32 14 93 73 90 7 69 84 92 28 31 58 60 79 55 **51** 71 17 35 23 83 87 25 **52 9** 50 95 81 13 5 63 18 16 21 3 96 27 59 44 76 22 97 11 10 70 4 85 20 37 0 26 19 29 67 68 99 24 53 1 43 86 72 75 94 **89**

Table 6: RND instances in OWL Manchester syntax. It is only shown the example 1 (avoiding school).

Terms	OWL Manchester syntax

AntennaSector1	`Individual:`
	` AntennaSector1`
	`Types:`
	` urban:Antenna`
	`Facts:`
	` urban:avoids School`

School	`Individual: School`
	`Types:`
	` geosparql:Building`
	`Facts:`
	` sao:hasLocation SchoolLocation`

SchoolLocation	`Individual:`
	` SchoolLocation`
	`Types:`
	` geosparql:Point`
	`Facts:`
	` urban:hasX "0"^^xsd:int,`
	` urban:hasY "0"^^xsd:int`

Our motivation has been driven by the rise of efforts to automatize the injection of domain knowledge during the optimization process, with the aim of allowing the semi-transparent setting of preferences (with semantic annotation) to a human expert in the problem domain, although without strong experience in optimization algorithms.

After the experimental validation, three main conclusions can be drawn:

1. Semantic technologies allow describing in a high level of abstraction knowledge domain in optimization problem modelling; consequently, users are able to indicate their requirements easily without needing to re-code algorithms.
2. The process of transforming the high level knowledge domain into problem constraints are done through mapping functions, although following general software design to allow coping with other different domains.
3. The proposed semantic model provides automatic mechanisms to inject specific problem constraints in the source code. This feature is a new approach to enable the map-

Table 7: Antennae locations obtained when approaching bi-objective RND problem using the user's preferences

Case Study	Antenna Location instance 1	Antenna Location instance 2
Avoid School (position 0)	**0** 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	**0** 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
Fixed Antenna Location (position 9)	1 1 0 1 1 1 1 1 **1** 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 **1** 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
Fixed Antenna Location by Client2 Location (position 25)	1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0 **1** 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 **1** 1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
50% minimum coverage	0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1
All preferences	**0** 0 0 1 1 1 1 0 0 **1** 1 1 1 1 0 1 1 1 1 1 1 0 1 0 **1** 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	**0** 0 0 0 0 0 0 0 0 **1** 0 1 0 0 0 0 0 0 1 0 1 0 0 1 **1** 1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0

ping between the concept ideas of the user's requirements and their implementations in the code.

4. The use cases based on well-known optimization problems in urban based knowledge domain have been useful to demonstrate the proposal in terms of proof-of-concept. It can be used as starting point for new emerging applications requiring the automatic modeling of optimization processes.

An open research line includes the definition of a richer ontology for the transportation domain, and to investigate a systematic approach to inject domain knowledge into the implementation of selection and variation (mutation and crossover) operators in evolutionary algorithms. Another important open research lies in the possibility of using the semantic knowledge-base, now consolidated in RDF, to perform reasoning tasks, hence to infer new knowledge. This could be useful not only for the formulation of additional implicit problem constraints, but also for the refinement of algorithmic operators specific to each problem instance.

Acknowledgements

This work has been partially funded by Spanish Grant TIN2017-86049-R (Spanish Ministry of Education and Science) and Andalusian PAIDI program with grant P18-RT-2799. José García-Nieto is the recipient of a Post-Doctoral fellowship of "Captación de Talento para la Investigación" Plan Propio at Universidad de Málaga.

References

[1] C. Blum, A. Roli, Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, ACM Computing Surveys 35 (3) (2003) 268–308 (2003).

[2] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: A survey of the state of the art, Swarm and Evolutionary Computation 1 (1) (2011) 32 – 49 (2011). doi:https://doi.org/10.1016/j.swevo.2011.03.001.
URL http://www.sciencedirect.com/science/article/pii/S2210650211000058

[3] A. J. Nebro, J. J. Durillo, J. García-Nieto, C. Barba-González, J. Del Ser, C. A. Coello Coello, A. Benítez-Hidalgo, J. F. Aldana-Montes, Extending the speed-constrained multi-objective pso (smpso) with reference point based preference articulation, in: A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, D. Whitley (Eds.), Parallel Problem Solving from Nature – PPSN XV, Springer International Publishing, Cham, 2018, pp. 298–310 (2018).

[4] K. Miettinen, Nonlinear multiobjective optimization, Vol. 12, Springer Science & Business Media, 2012 (2012).

[5] A. Jaszkiewicz, J. Branke, Interactive multiobjective evolutionary algorithms, in: Multiobjective optimization, Springer, 2008, pp. 179–193 (2008).

[6] H. Kaiya, M. Saeki, Using domain ontology as domain knowledge for requirements elicitation, in: 14th IEEE International Requirements Engineering Conference (RE'06), IEEE, 2006, pp. 189–198 (2006).

[7] N. F. Noy, D. L. McGuinness, Ontology development 101: A guide to creating your first ontology (2001).

[8] N. Guarino, Formal ontology and information systems, in: Proceedings of FOIS, Vol. 98, 1998, pp. 81–97 (1998).

[9] M. del Mar Roldán-García, M. J. García-Godoy, J. F. Aldana-Montes, Dione: An owl representation of icd-10-cm for classifying patients' diseases, Journal of biomedical semantics 7 (1) (2016) 62 (2016).

[10] M. Hülsen, J. M. Zöllner, C. Weiss, Traffic intersection situation description ontology for advanced driver assistance, in: Intelligent Vehicles Symposium (IV), 2011 IEEE, IEEE, 2011, pp. 993–999 (2011).

[11] R. Mizoguchi, M. Ikeda, Towards ontology engineering, Journal-Japanese Society for Artificial Intelligence 13 (1998) 9–10 (1998).

[12] A. Yaman, A. Hallawa, M. Coler, G. Iacca, Presenting the ECO: Evolutionary computation ontology, in: European Conference on the Applications of Evolutionary Computation, 2017, pp. 603–619 (2017).

[13] L. Li, I. Yevseyeva, V. Basto-Fernandes, H. Trautmann, N. Jing, M. Emmerich, Building and using an ontology of preference-based multiobjective evolutionary algorithms, in: H. Trautmann, G. Rudolph, K. Klamroth, O. Schütze, M. Wiecek, Y. Jin, C. Grimme (Eds.), Evolutionary Multi-Criterion Optimization: 9th International Conference, EMO 2017, Münster, Germany, March 19-22, 2017, Proceedings, Springer International Publishing, Cham, 2017, pp. 406–421 (2017).

[14] C. Barba-González, J. García-Nieto, M. del Mar Roldán-García, I. Navas-Delgado, A. J. Nebro, J. F. Aldana-Montes, Bigowl: Knowledge centered big data analytics, Expert Systems with Applications 115 (2019) 543–556 (2019).

[15] N. Guarino, Formal ontology, conceptual analysis and knowledge representation, International journal of human-computer studies 43 (5-6) (1995) 625–640 (1995).

[16] C. Brewster, K. O'Hara, Knowledge representation with ontologies: the present and future, IEEE Intelligent Systems 19 (1) (2004) 72–81 (2004).

[17] P. P. Bonissone, R. Subbu, N. Eklund, T. R. Kiehl, Evolutionary algorithms + domain knowledge = real-world evolutionary computation, IEEE Transactions on Evolutionary Computation 10 (3) (2006) 256–280 (June 2006).

[18] J. Durillo, A. Nebro, jmetal: A java framework for multi-objective optimization, Advances in Engineering Software 42 (10) (2011) 760 – 771 (2011). doi:10.1016/j.advengsoft.2011.05.014.

[19] A. Nebro, J. J. Durillo, M. Vergne, Redesigning the jMetal multi-objective optimization framework, in: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO Companion '15, ACM, New York, NY, USA, 2015, pp. 1093–1100 (2015). doi:10.1145/2739482.2768462.

[20] P. Bellini, M. Benigni, R. Billero, P. Nesi, N. Rauch, Km4city ontology building vs data harvesting and cleaning for smart-city services, Journal of Visual Languages & Computing 25 (6) (2014) 827–839 (2014).

[21] R. Battle, D. Kolas, Geosparql: enabling a geospatial semantic web, Semantic Web Journal 3 (4) (2011) 355–370 (2011).

[22] N. Komninos, C. Bratsas, C. Kakderi, P. Tsarchopoulos, Smart city ontologies: Improving the effectiveness of smart city applications, Journal of Smart Cities 1 (1) (2015) 31–46 (2015).

[23] C. Barba-González, J. García-Nieto, A. J. Nebro, J. A. Cordero, J. J. Durillo, I. Navas-Delgado, J. F. Aldana-Montes, jmetalsp: A framework for dynamic multi-objective big data optimization, Applied Soft Computing 69 (2018) 737 – 748 (2018). doi:https://doi.org/10.1016/j.asoc.2017.05.004.
URL http://www.sciencedirect.com/science/article/pii/S1568494617302557

[24] S. P. Mendes, G. Molina, M. A. Vega-Rodriguez, J. A. Gomez-Pulido, Y. Saez, G. Miranda, C. Segura, E. Alba, P. Isasi, C. Leon, J. M. Sanchez-Perez, Benchmarking a wide spectrum of metaheuristic techniques for the radio network design problem, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 1133–1150 (Oct 2009). doi:10.1109/TEVC.2009.2023448.

[25] M. Horridge, N. Drummond, J. Goodwin, A. L. Rector, R. Stevens, H. Wang, The manchester owl syntax., in: OWLed, Vol. 216, 2006 (2006).

[26] T. Weise, M. Zapf, R. Chiong, A. J. Nebro, Why Is Optimization Difficult?, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 1–50 (2009). doi:10.1007/978-3-642-00267-0_1.
URL https://doi.org/10.1007/978-3-642-00267-0_1

[27] R. Krummenacher, T. Strang, Ontology-based context modeling, in: In Workshop on Context-Aware Proactive Systems, 2007 (2007).

[28] E. Blomqvist, K. Sandkuhl, Patterns in ontology engineering: Classification of ontology patterns., in: ICEIS (3), 2005, pp. 413–416 (2005).

[29] B. McBride, The resource description framework (rdf) and its vocabulary description language rdfs, in: Handbook on ontologies, Springer, 2004, pp. 51–65 (2004).

[30] T. R. Gruber, A translation approach to portable ontology specifications, Knowledge acquisition 5 (2) (1993) 199–220 (1993).

[31] R. Uceda-Sosa, B. Srivastava, R. J. Schloss, Building a highly consumable semantic model for smarter cities, in: Proceedings of the AI for an Intelligent Planet, ACM, 2011, p. 3 (2011).

[32] R. Troncy, G. Rizzo, A. Jameson, O. Corcho, J. Plu, E. Palumbo, J. C. B. Hermida, A. Spirescu, K.-D. Kuhn, C. Barbu, et al., 3cixty: Building comprehensive knowledge bases for city exploration, Journal of Web Semantics 46 (2017) 2–13 (2017).

[33] P. Espinoza-Arias, M. Poveda-Villalón, R. García-Castro, O. Corcho, Ontological representation of smart city data: From devices to cities, Applied Sciences 9 (1) (2019) 32 (2019).

[34] S. Staab, R. Studer, Handbook on ontologies, Springer Science & Business Media, 2010 (2010).

[35] N. Noy, D. M. (Hrsg.), Ontology development 101: A guide to creating your first ontology. Technical report, 2001 (2001).

[36] D. Puiu, P. Barnaghi, R. Tönjes, D. Kümper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. Farajidavar, et al., Citypulse: Large scale data analytics framework for smart cities, IEEE Access 4 (2016) 1086–1108 (2016).

[37] M. S. Fox, The semantics of populations: A city indicator perspective, Journal of Web Semantics 48 (2018) 48–65 (2018).

[38] M. Horridge, S. Bechhofer, The owl api: A java api for owl ontologies, Semantic Web 2 (1) (2011) 11–21 (2011).

[39] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197 (2002).

[40] G. Reinelt, TSPLIB–a traveling salesman problem library, ORSA Journal on Computing 3 (4) (1991) 376–384 (1991).

[41] A. Nebro, E. Alba, G. Molina, F.Luna, J. Durillo, Optimal antenna placement using a new multi-objective chc algorithm, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, 2007, pp. 876–883 (2007).

[42] A. J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, J. J. Durillo, Optimal antenna placement using a new multi-objective chc algorithm, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, pp. 876–883 (2007).

[43] P. Moscato, C. Cotta, A Gentle Introduction to Memetic Algorithms, Springer US, Boston, MA, 2003, pp. 105–144 (2003). doi:10.1007/0-306-48056-5_5.
URL https://doi.org/10.1007/0-306-48056-5_5

Appendix

Table 8: TSP instances of fixed position examples in OWL Manchester syntax.

Terms	OWL Manchester syntax
City	Individual: City Types: geosparql:City Facts: urban:contains BakeryClient, urban:contains FirstQuarterClient, urban:contains Client2, urban:contains Client3, urban:contains RestaurantClient, urban:contains School, urban:contains VIP, urban:hasGrid CityGrid, urban:hasRoute DeliveryRoute
Subroute	Individual: DeliveryRoute Types: urban:Route Facts: urban:hasPreference FirstQuarterPreference, urban:hasPreference HalfWayClientPreference, urban:hasPreference LastClientPreference, urban:hasPreference SubroutePreference, urban:hasPreference VIPPreference, urban:hasLength "100"^^xsd:int
VIPPreference	Individual: VIPPreference Types: urban:Preference Facts: urban:hasPositionInPreference VIPPosition
VIPPosition	Individual: VIPPosition Types: urban:Position Facts: urban:hasCityObject VIP, bigowl:hasDataType "integer"^^xsd:string, urban:hasPositionValue "1"^^xsd:int,
VIP	Individual: VIP Types: geosparql:Building Facts: sao:hasLocation VIPLocation, urban:hasId "78"^^xsd:int
VIPLocation	Individual: VIPLocation Types: geosparql:Point Facts: urban:hasX "10"^^xsd:int, urban:hasY "20"^^xsd:int
HalfWayClientPreference	Individual: HalfWayClientPreference Types: urban:Preference Facts: urban:hasPositionInPreference HalfWayPosition
HalfWayPosition	Individual: HalfWayPosition Types: urban:Position Facts: urban:hasCityObject Client2, bigowl:hasDataType "integer"^^xsd:string, urban:hasPositionValue "50"^^xsd:int
Client2	Individual: Client2 Types: geosparql:Building Facts: sao:hasLocation Client2Location, urban:hasId "55"^^xsd:int
Client2Location	Individual: Client2Location Types: geosparql:Point Facts: urban:hasX "2"^^xsd:int, urban:hasY "5"^^xsd:int
LastClientPreference	Individual: LastClientPreference Types: urban:Preference Facts: urban:hasPositionInPreference LastClientPosition
LastClientPosition	Individual: LastClientPosition Types: urban:Position Facts: urban:hasCityObject Client100, bigowl:hasDataType "string"^^xsd:string, urban:hasPositionValue "100"^^xsd:int
Client100	Individual: Client100 Types: geosparql:Building Facts: sao:hasLocation Client100Location, urban:hasId "89"^^xsd:int
Client100Location	Individual: Client100Location Types: geosparql:Point Facts: urban:hasX "5"^^xsd:int, urban:hasY "5"^^xsd:int

Table 9: TSP instances of variable position examples in OWL Manchester syntax.

Terms	OWL Manchester syntax
FirstQuarterPreference	Individual: FirstQuarterPreference Types: urban:Preference Facts: urban:hasPositionInPreference FirstQuarterPosition
FirstQuarterPosition	Individual: FirstQuarterPosition Types: urban:Position Facts: urban:hasCityObject Client1, bigowl:hasDataType "string"^^xsd:string, urban:hasPositionValue "<=25%"^^xsd:string
Client1	Individual: Client1 Types: geosparql:Building Facts: sao:hasLocation Client1Location, urban:hasId "49"^^xsd:int
Client1Location	Individual: Client1Location Types: geosparql:Point Facts: urban:hasX "1"^^xsd:int, urban:hasY "9"^^xsd:int
SubroutePreference	Individual: SubroutePreference Types: urban:Preference Facts: urban:hasPositionInPreference BakeryPosition, urban:hasPositionInPreference RestaurantPosition
BakeryPosition	Individual: BakeryPosition Types: urban:Position Facts: urban:hasCityObject Bakery, urban:hasNext RestaurantPosition, urban:isFirst true
Bakery	Individual: Bakery Types: geosparql:Building Facts: sao:hasLocation BakeryLocation, urban:hasId "52"^^xsd:int
BakeryLocation	Individual: BakeryLocation Types: geosparql:Point Facts: urban:hasX "2"^^xsd:int, urban:hasY "4"^^xsd:int
RestaurantPosition	Individual: RestaurantPosition Types: urban:Position Facts: urban:hasCityObject Restaurant
Restaurant	Individual: Restaurant Types: geosparql:Building Facts: urban:hasId "9"^^xsd:int
RestaurantLocation	Individual: RestaurantLocation Types: geosparql:Point Facts: urban:hasX "3"^^xsd:int, urban:hasY "3"^^xsd:int

Table 10: RND instances in OWL Manchester syntax.

Terms	OWL Manchester syntax
CityGrid	Individual: CityGrid Types: urban:Grid Facts: urban:hasGridConstraint CityGridConstraints, urban:hasGridSize "1000"^^xsd:int
CityGridConstraints	Individual: CityGridConstraints Types: urban:GridConstraint Facts: urban:hasAntenna AntennaSector1, urban:hasAntenna AntennaSector2, urban:hasAntenna AntennaSector3, urban:hasMinimumCoverage "50.0"^^xsd:double
AntennaSector1	Individual: AntennaSector1 Types: urban:Antenna Facts: urban:avoids School
School	Individual: School Types: geosparql:Building Facts: sao:hasLocation SchoolLocation
SchoolLocation	Individual: SchoolLocation Types: geosparql:Point Facts: urban:hasX "0"^^xsd:int, urban:hasY "0"^^xsd:int
AntennaSector2	Individual: AntennaSector2 Types: urban:Antenna Facts: seo:hasLocation AntennaSector2Location
AntennaSector2Location	Individual: AntennaSector2Location Types: geosparql:Point Facts: urban:hasX "1"^^xsd:int, urban:hasY "9"^^xsd:int
AntennaSector3	Individual: AntennaSector3 Types: urban:Antenna Facts: urban:attaches Client2
Client2	Individual: Client2 Types: geosparql:Building Facts: sao:hasLocation Client2Location, urban:hasId "55"^^xsd:int
Client2Location	Individual: Client2Location Types: geosparql:Point Facts: urban:hasX "9"^^xsd:int, urban:hasY "9"^^xsd:int