International Journal of Geo-Information

MDPI

Article

Multi-Agent Planning for Automatic Geospatial Web Service Composition in Geoportals

Mahdi Farnaghi [1,2,*] and Ali Mansourian [1,3]

1 GIS Center, Department of Physical Geography and Ecosystem Science, Lund University, 22362 Lund, Sweden; ali.mansourian@nateko.lu.se
2 Faculty of Geodesy and Geomatics Engineering, K. N. Toosi University of Technology, Tehran 19697 64499, Iran
3 Center for Middle-Eastern Studies, Lund University, 22362 Lund, Sweden
* Correspondence: mahdi.farnaghi@nateko.lu.se or farnaghi@kntu.ac.ir; Tel.: +46-46-222-17-33

Received: 4 September 2018; Accepted: 9 October 2018; Published: 12 October 2018

Abstract: Automatic composition of geospatial web services increases the possibility of taking full advantage of spatial data and processing capabilities that have been published over the internet. In this paper, a multi-agent artificial intelligence (AI) planning solution was proposed, which works within the geoportal architecture and enables the geoportal to compose semantically annotated Open Geospatial Consortium (OGC) Web Services based on users' requirements. In this solution, the registered Catalogue Service for Web (CSW) services in the geoportal along with a composition coordinator component interact together to synthesize Open Geospatial Consortium Web Services (OWSs) and generate the composition workflow. A prototype geoportal was developed, a case study of evacuation sheltering was implemented to illustrate the functionality of the algorithm, and a simulation environment, including one hundred simulated OWSs and five CSW services, was used to test the performance of the solution in a more complex circumstance. The prototype geoportal was able to generate the composite web service, based on the requested goals of the user. Additionally, in the simulation environment, while the execution time of the composition with two CSW service nodes was 20 s, the addition of new CSW nodes reduced the composition time exponentially, so that with five CSW nodes the execution time reduced to 0.3 s. Results showed that due to the utilization of the computational power of CSW services, the solution was fast, horizontally scalable, and less vulnerable to the exponential growth in the search space of the AI planning problem.

Keywords: multi-agent artificial intelligence (AI) planning; automatic web service composition; OGC web service; semantic web; geoportal

1. Introduction

During the last decade, an enormous amount of spatial information and processing capabilities have been published over the internet. To meet the interoperability of the systems, Open Geospatial Consortium Web Services (OWS) have been used as the underlying technology. The need to integrate atomic OWS nodes into complex workflows is an important milestone on the path to fully benefit from published spatial data and functionality over the internet. This integration of OWSs, known as web service composition, provides the geospatial community with a flexible opportunity to address more complicated analytical requirements using distributed spatial information and knowledge organization systems.

Web service composition is defined as the process of creating web service chains and establishing new functionalities by composing a collection of web services [1]. Web service composition aims to arrange several web services into one complex service to achieve complex objectives [2–4].

Discovery and selection of proper OWSs and combining them into a new composite web service is a complex and time-consuming task, which requires experts with different proficiencies to handle it manually. Automatic generation of composite OWSs is a promising alternative that can significantly reduce the complexity of integrating OWSs. Using the automatic composition techniques, users formalize their requirements as a goal and send it to a composer component. The composer component utilizes the syntactic and semantic description of OWSs to generate a new workflow to address the user's goal.

A few researchers in the geospatial community, including Yue et al. [5], Cruz et al. [6], Farnaghi and Mansourian [7], Farnaghi and Mansourian [8], and Al-Areqi et al. [9] have tried to solve the problem of automatic composition of OWSs. The overall approach in their studies was to semantically or syntactically describe specific characteristics of OWSs, and then exploit centralized artificial intelligence (AI) planning techniques [10] to generate an execution workflow from atomic OWSs. However, this approach encounters two critical challenges: Incompatibility with the distributed architecture of geoportals and exponential growth of the search space.

The first challenge is that centralized AI planning techniques, which have been used in previous studies for automatic composition of OWSs, are not compatible with the distributed architecture of geoportals [11,12], which have been developed and used over the last fifteen years in the geospatial community for the search and discovery of spatial data and OWSs. Based on the standard architecture of geoportals, OWSs are registered in Catalogue Service for Web (CSW) services [12]. Using the geoportal interface, users can specify the characteristics of their desired geospatial data or services. The geoportal then executes a distributed search over the registered CSW services and matching results are presented to the user. However, it is a challenging task to use a centralized planning technique for automatic composition in geoportals. The critical point is that, to be able to solve the composition problem, the centralized AI planning techniques require accessing all available OWSs simultaneously. Nevertheless, in geoportals, OWSs are registered in CSW services. To solve this problem, one may send a loose search condition to every CSW service, retrieve required information about all available OWSs from every CSW service, and then try to solve the composition problem on a single computational node. However, retrieving all available OWSs and trying to solve an AI planning problem with a massive amount of OWSs leads to a second challenge of the exponential growth of the search space as follows.

Although the AI planning techniques have proved to be the most promising solution for automatic composition of web services (see Reference [13] for a detailed comparison of web service composition techniques) and hence the automatic composition of OWSs, they still encounter serious problems in terms of dealing with real-world scenarios. Considering the huge amount of published OWSs over the internet, centralized AI planning techniques in which a planner agent is responsible for solving the whole planning problem alone, are not efficient and feasible. That's mainly because, as the number of available OWSs increases, the search space of the AI planning grows exponentially (see References [14–16]). Therefore, in real-world scenarios the centralized planning techniques are not able to solve the automatic composition problem in an acceptable time.

To address these two mentioned problems, this article presents a novel solution for automatic composition of OWSs. The solution proposes a new multi-agent artificial intelligence (AI) planning algorithm in which OWSs are semantically annotated and registered in CSW services (Catalogue Service for Web, [12]). The CSW services are considered as agents that work in a multi-agent system. Unlike the centralized algorithms, the proposed algorithm utilizes the computational powers of the distributed CSW services to solve the composition problem collaboratively. The solution is an extension to the distributed architecture of geoportals [17–19]. In addition to being fast and scalable, this solution is expected to be less vulnerable to the exponential growth in the search space of the composition problem. The feasibility of the algorithm is demonstrated using a case study and a simulated testbed of one hundred OWSs.

The remainder of this article is organized as follows. In Section 2, related studies on the automatic composition of OWSs and other web service types are described, and current challenges in automatic

web service composition are highlighted. The proposed solution for automatic composition of OWSs is thoroughly described in Section 3. Section 4 is dedicated to implementation and demonstration of the results. Finally, Section 5 provides a discussion and Section 6 concludes the paper.

2. Related Studies

In the geospatial community, most studies have been concentrated on the manual or semi-automatic composition of OWSs (see References [20–28]). However, automatic generation of composite OWSs has received significant attention in recent years. Yue et al. [5] have utilized a HTN (Hierarchical Task Network) planning technique to generate composite geospatial web services out of semantically described OWSs. OWL-S (Semantic Markup for Web Services, [29]) was used as an underlying technology to describe semantic aspects of OWSs. In another study, Cruz, Monteiro, and Santos [6] presented a solution for the automatic composition of OWSs based on geodata quality. The OWSs were semantically described by OWL-S and a backward planning algorithm was used to generate the composite web service. Farnaghi and Mansourian [7] reported a study in which OWSs were semantically described by a WSMO (Web Service Modeling Ontology, [30]) framework, and the composition process was conducted using the Graphplan algorithm. In another study, Farnaghi and Mansourian [8] proposed a comprehensive method for annotating OWSs using SAWSDL and WSMO-Lite (Lightweight Semantic Descriptions for Services on the Web, [31]), and then used a planning algorithm to generate composite OWSs. Du et al. [32] presented an architecture for the model-driven composition of OWSs, where the quality of service (QoS) was considered as an affecting factor. Al-Areqi, Lamprecht, and Margaria [9] reported a constraint-driven composition solution based on a web service modeling tool called PROPHETS. The tool synthesizes web services based on a search algorithm by evaluating constraints defined in SLTL (Semantic Linear Time Logic). However, to the best of the authors' knowledge, none of the proposed solutions for automatic composition of OWSs in the geospatial community were compatible with the distributed architecture of geoportals. Instead, they are centralized and do not have any solution for dealing with the huge amount of OWSs published over the internet.

In the information technology community, few studies have used multi-agent systems for the automatic composition of web services to overcome the above-mentioned issue. Vaithiyanathan and Govindharajan [33] proposed a system in which two agents cooperated with each other to generate composite W3C (World Wide Web Consortium) web services. However, in their system, the composition is done by one of the agents and the other agent is just responsible for the search and discovery of available web services. Therefore, the processing overload of the web service composition is not delegated to multiple agents. In two other studies, Küngas and Matskin [34] and Charif and Sabouret [35] separately developed multi-agent systems to enable dynamic composition and execution of W3C web services. Their systems were designed to dynamically execute the composition, which meant that the output compositions were not registered in an orchestration engine, and hence were not reusable in future. El Falou et al. [15] presented a distributed, multi-agent algorithm for automatic generation of composite W3C web services. They utilized a multi-agent AI planning algorithm to generate the composition. However, they focused on the syntactic aspects of W3C web services, and semantic description of web services was not used in the composition algorithms. These studies in the information technology community do not conform to the standards of OWSs that are generally accepted in GIS communities, and therefore cannot be used for automatic composition of OWSs.

3. Method

The conventional geoportals, based on the OGC (Open Geospatial Consortium) geoportal architecture [11,36], leave users alone when there is no geospatial data or service that can satisfy the users' requirements [37]. To improve the functionality of the conventional geoportals with the ability to automatically compose OWSs, this study proposes a solution based on multi-agent AI planning techniques. Multi-agent AI planning is a coordination mechanism [38–41] wherein to coordinate

agents' actions and avoid conflicts, a plan consisting of actions of all participating agents is generated, so that execution of that plan leads to the satisfaction of the global goal of the system along with the personal goal of every agent (for a detailed review of multi-agent AI planning techniques see Reference [41]). The proposed solution in this study considers a geoportal as a collection of agents that are working together with the same goal of satisfying the user's need. Where the system cannot find a proper OWS to address a user's need, it tries to generate a composite web service through the proposed multi-agent AI planning algorithm.

To thoroughly describe the proposed solution, the components of the system are briefly described, first, from an architectural point of view in Section 3.1. Then, the semantic annotation mechanism which has been adapted to describe different aspects of OWSs is reviewed in Section 3.2. To define a proper notation for an in-depth description of the proposed composition algorithm, a conceptual framework is defined afterward in Section 3.3. Finally, using the conceptual model, the AI planning algorithm is comprehensively elaborated in Section 3.4.

3.1. Architecture

With the purpose of adding the ability of automatic OWS composition to geoportals, this study suggested that three components of an OWS composition interface, an orchestration engine, and a composition coordinator component should be added to the typical architecture of geoportals (Figure 1). Additionally, the CSW services should be improved to be able to participate in the proposed multi-agent AI planning algorithm.

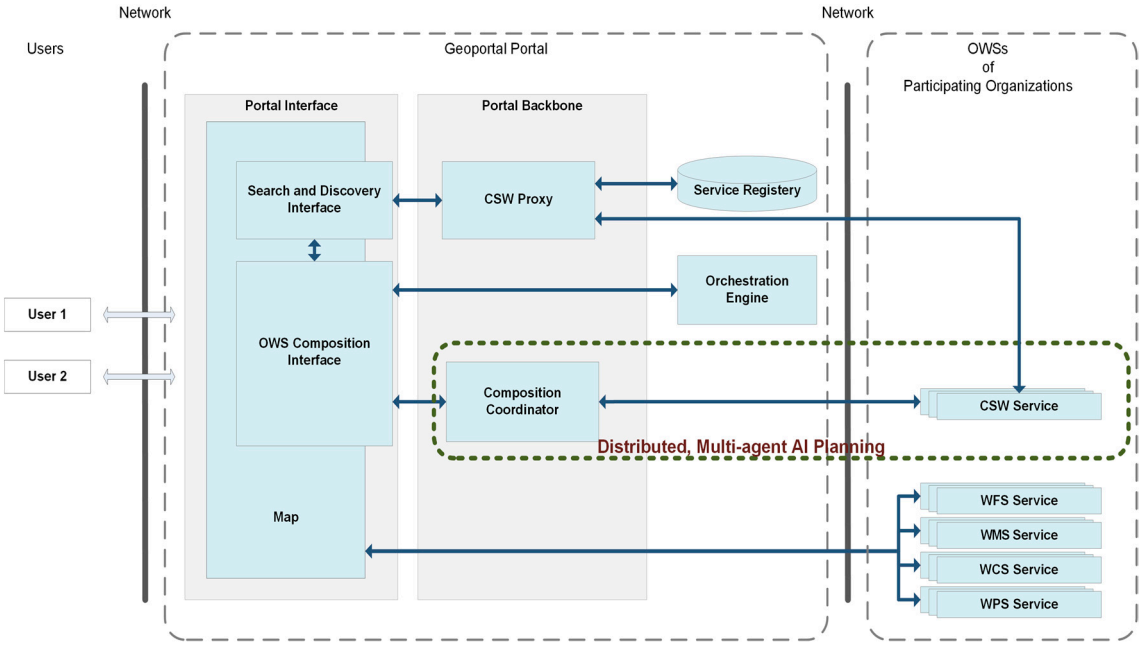

Figure 1. Geoportal architecture with automatic composition capabilities.

In the architecture depicted in Figure 1, like any conventional geoportal, OWSs are registered in CSW services. Using the search and discovery interface, users can specify the characteristics of the desired geospatial data or services. The interface then executes a distributed search through the CSW proxy component over the registered CSW services. If the search is successful, matching data and services are presented to the user through the search and discovery interface, and in some cases, the user can also visualize the output data or services on the map.

Where users cannot find proper data or service to satisfy their needs, they can use the composition capabilities of the system. Using the OWS composition interface, an expert user will be able to manually combine a list of selected OWSs, generate a composition, and register it into the orchestration

engine, so that the new composite web service can be executed. This architecture also enables the user to generate composite OWSs automatically. In this regard, the user sends their requirements as a composition goal to the composition coordinator component. The composition coordinator component commences the proposed multi-agent AI planning algorithm (see Section 3.4) to automatically generate a composition that satisfies the user's goal. The output composite web service is then registered in the orchestration engine as a new OWS.

3.2. Semantic Annotation of OWSs

Automatic generation of composite OWSs requires that a software component first locates the available OWSs and evaluates their characteristics. To address these requirements, the OWSs have to be described in a machine-readable and machine-understandable way [42]. This can be realized through a syntactic and semantic description of OWSs. While the OGC implementation specifications (see References [43–47]) address the syntactic aspects of OWSs, the semantic annotation can be used to provide the semantic description.

In this study, an approach based on WSDL (Web Service Definition Language), SAWSDL (Semantic Annotation for WSDL, [48]) and SPARQL (pronounced "sparkle", a recursive acronym for SPARQL Protocol and RDF Query Language) [49], proposed by Farnaghi and Mansourian [8], was adapted for semantic annotation of OWSs. Contrary to comprehensive and demanding top-down approaches like OWL-S [50], SWSF [51], and WSMO [30], which assume the semantic description of a web service is developed in parallel with the web service itself, the selected semantic annotation approach is bottom-up, light-weight, and agile. Therefore, it is well suited for semantic annotation of available services that are already implemented syntactically [8].

Based on the semantic annotation approach (Figure 2), the capability document of an OWS is supported by a WSDL document, where service, interface, operations, and types of the OWS are annotated through links to their respective semantic concepts, and lowering and lifting transformations using SAWSDL. SAWSDL is a W3C recommendation standard, which supports adding semantic annotations to WSDL documents. Using SAWSDL, the service element in the WSDL points to ontological concepts in non-functional ontology and OWS classification ontology. The operations in the WSDL document are marked by operation types in OWS classification ontology. Additionally, the conditions upon which an operation can be executed and the information that will be added after execution of the operation are defined by SPARQL [49] ask and update queries (see Section 3.3) as condition and effect, respectively, and linked to the operation. The SPARQL queries are defined based on the service ontology. The types that are used in the definition of operations in the WSDL document are also annotated by ontological concepts defined in service ontology.

The conversion between the semantic and syntactic representation of each type is defined using lowering and lifting transformations. Elements of service ontology, OWS classification ontology, and non-functional ontology are all defined based on classes and properties from WSMO-Lite ontology [31] and WSMO-Lite Extension ontology (for definition of condition, effect, functional classification, and non-functional parameter); GML Coverage Profile ontology and GML Simple Feature Profile ontology (for definition of geographical concepts, e.g., point, polyline, polygon, etc.); and COSMO (COmmon Semantic Model, available at http://micra.com/COSMO/COSMO.owl) top-level ontology and COSMO Extension ontology (for refereeing to real-world objects and phenomena, e.g., river, mountain, land, etc.). The COSMO ontology was selected due to its ability to provide broad semantic interoperability by integrating concepts from other foundation ontologies including OpenCyc, SUMO, DOLCE, and BFO. MSO and OntoMap are the two other foundation ontologies that could have been used in this sense.

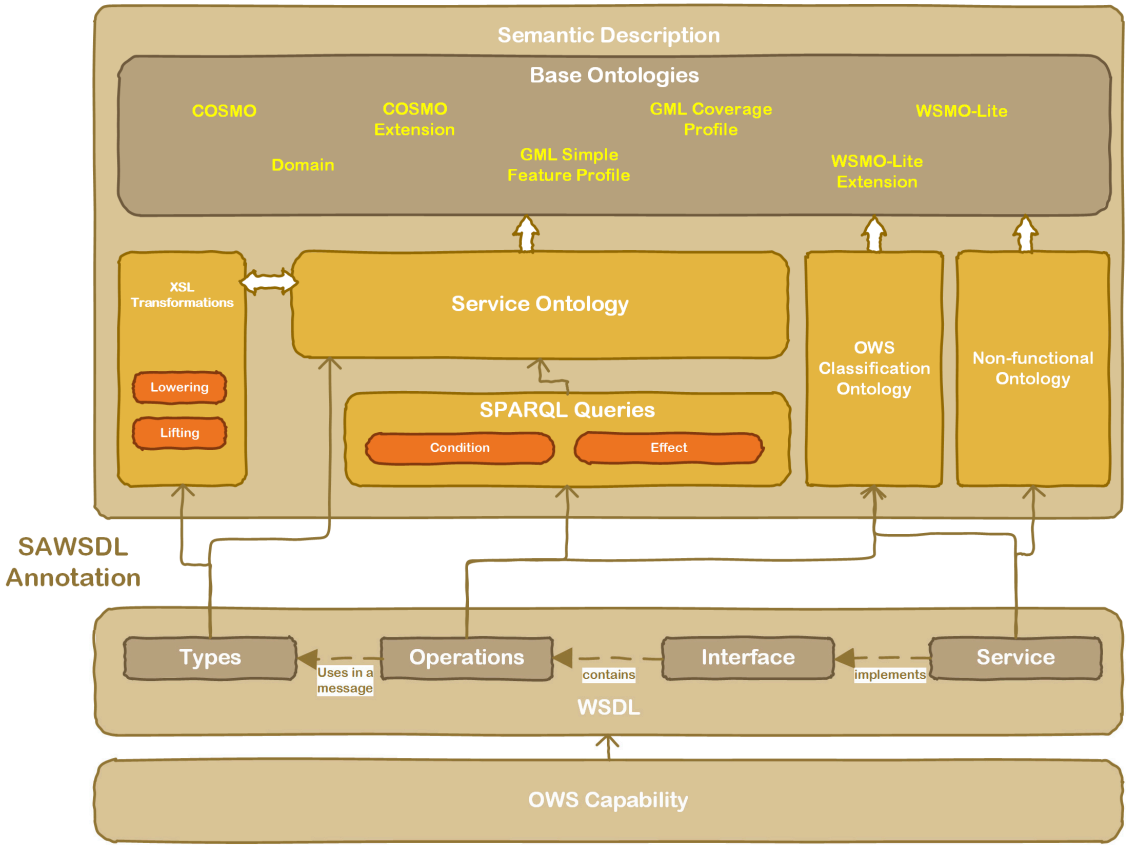

Figure 2. Semantic Annotation of Open Geospatial Consortium Web Services (OWSs), (adapted from Farnaghi and Mansourian [8] and modified).

3.3. A Conceptual Model for Automatic Composition of OWSs

To formally describe the proposed automatic composition algorithm, a conceptual model is presented that defines different roles of the participating elements of the algorithm. The model utilizes RDF (Resource Description Framework) language for representation of states in AI planning and SPARQL ask and update queries for condition checking and state transitions. The conceptual model supposes that several OWSs are registered in each CSW service. Each OWS is semantically annotated (see Section 3.2) and CSW services are accessible by the composition coordinator component. The formal representation of the model is as follows.

- $R = [r_0, r_1, r_2, \ldots]$ is a set of possible states. Each state $r_i \in R$ is represented as an RDF graph.
- q is a *SPARQL ask query* composed of one or more triple patterns that are connected by conjunction and disjunction operators.
- $\zeta(q, r)$ receives a SPARQL ask query, q, and an RDF graph representing a particular state, r. This function executes q on r and returns *true* if there exists at least a match for the query q in the graph r and $false$ if there is not.
- $\varepsilon(u, r)$ executes a *SPARQL update operation*, u, on an RDF graph representing a particular state, r, and returns the updated RDF graph, representing the new state.
- $r_0 \in R$ is the initial state.
- g is a *SPARQL ask query* that represents the composition goal.
- $CNF(q)$ is a function that converts a SPARQL ask query, q, to a conjunctive normal form (CNF) (see References [52,53]), where the query is represented as a conjunction of triple patterns.
- a is an operation of an OWS service. Since OWSs are semantically annotated, each operation is defined as $a = [condition, effect]$.

  - $a.condition$, is a *SPARQL ask query* that determines whether a is applicable to a particular state.
  - $a.effect$, is a *SPARQL update query* that inserts some triples to the RDF graph of the underlying state. The added triples describe the information that will be added after the execution of the operation.

- Given an operation a, and a state, $r \in R$, a is applicable to r if $\zeta(a.condition, r)$ returns *true*. Applying a on r will be done by the execution of $a.effect$ on r and will result in a new state $r_{new} = \varepsilon(a.effect, r)$.
- $\pi = A^1 \succ A^2 \succ \ldots \succ A^k \Big| A^i = \{a_1, \ldots, a_{ni}\}$ is a partial-ordered plan, represented as an ordered sequence of sets of OWSs operations. Each set is composed of OWS operations.
- $\mathrm{E}(\pi, r)$ executes a partial-ordered plan, π, on a particular state, r, and generates a result state. π can be executed on r if all operations in its first set are applicable to r. Execution starts from the first set. All operations in the first set are executed on r using $\varepsilon(a.effect,\ r)$ and a result state is generated. Then the operations of the second set are executed on the result state of the first set. This sequetional execution of operations of a set on the result state of the previous set continues to the last set in π. π is a valid partial-ordered plan, if every operation in each set is executable on the result state of the previous set.
- Solving the automatic composition problem is to find a valid partial-ordered plan, where its execution on the initial state, r_0, results in a final state, r_f, so that the goal, g, is satisfied and $\zeta\left(g,\ r_f\right) = true$.
- *cc* is the composition coordinator component. This component implements the following methods:

  - $GenerateMultiAgentPlan(r,\ g)$ receives an initial state, r, a goal, g, and returns a valid partial-ordered plan, if there is any, as the result of the multi-agent AI planning algorithm.
  - $GetRegisteredCswAgents()$ returns the list of registered CSW services as $\left[CA^1,\ CA^2,\ \ldots\right]$.

- CA^i is a CSW service, which in addition to the standard operations of CSW services [12] implements the following methods:

  - $GetRegisteredOWSOperators()$ returns a list of every operation of OWSs which are registered in CA^i in the form of $A^i = \left[a_1^i,\ a_2^i,\ \ldots\right]$. These operations are considered as actions of the AI planning model.
  - $GenerateExecutablePartialOrderedPlans(r_c)$ is a method that receives a state, r_c, and the goal, g, and tries to find every possible executable partial-ordered plan composed of the operations in A^i and are executable on r_c.

Based on this conceptual model, Section 3.4 describes the automatic composition algorithm.

3.4. Automatic Composition Algorithm

The proposed algorithm is a distributed descendant of A* heuristic search algorithm. As a generic, best-first search algorithm, A* tries to solve a problem by searching among all possible paths to the goal, where the search is guided through a heuristic function that calculates cost of every path in the graph and forces the algorithm to pursue the path with minimum cost that appears to lead most quickly to the goal.

The proposed algorithm perceives the composition as a pathfinding problem in a graph, where states are the nodes and partial-ordered plans are the links. The algorithm runs in a multi-agent environment where a composition coordinator component directs the search within the graph, and multiple CSW agents are responsible for expanding the search graph. None of the agents has complete knowledge of the search space, each CSW agent only knows the operations of their OWSs, and the composition coordinator agent has no knowledge of OWSs registered in the CSW services.

3.4.1. Composition Coordinator Component

The automatic composition algorithm starts when the *Generate Multi Agent Plan* method of the composition coordinator component is called. Figure 3 shows the corresponding pseudo code. The algorithm receives the initial state r_0, and the composition goal, g, as inputs, and tries to generate a plan that can satisfy g. The composition coordinator component first retrieves the list of all registered CSW services by calling the *Get Registered Csw Agents* method. Then it generates an initial search node *initNode*, with an empty plan, state r_0, and infinitive cost. It creates an open set *openSet*, which will contain the search nodes that have not yet been expanded, and a closed set, *closedSet*, which includes all the search nodes that have been recently expanded (lines 1 to 5, Figure 3). In this step, the open set only includes the initial state and the closed set is empty. The algorithm starts a loop which continues whilst the open set is not empty (lines 6 to 24, Figure 3). Within the loop, the algorithm first calculates the cost for every node in the open set using $f(node,\ g) = c(node) + h(node,\ g)$, where:

- $c(node)$ calculates the cost of reaching from the initial state to a state of interest, $node.State$. In the composition, it is expected to reach the goal state by applying a minimum number of OWS operations. To model this condition, in Equation (1), Sigmoid function is applied on the number of operations that compose the plan of the node. Sigmoid function was used, so that the output value is between 0 and 1.
- $h(node,\ g)$ is a heuristic function that estimates the cost to go from $node.State$ to the goal state, g. When the state of a search node has satisfied more RDF triples in the SPARQL ask query of the goal, it is more likely that expansion of that node will lead the algorithm to the final state. Therefore, Equation (2) counts the number of triples in the CNF form of g that are not satisfied in $node.State$ and divide it by the total number of triples in the CNF form of g.

$$c(node) = \frac{1}{1 + e^{-(\#\ operations\ in\ node.Plan - x_0)}} \tag{1}$$

$$h(node,\ g) = \frac{\#\ triples\ of\ CNF(g)\ not\ satisfied\ by\ state\ node.State}{\#\ triples\ of\ CNF(g)} \tag{2}$$

```
1   GenerateMultiAgentPlan(r0, g)
2       cswAgents ← GetRegisteredCswAgents()
3       initNode ← [Plan: ⟨∅⟩, State: r0, Cost: ∞, Parent: ∅]
4       openSet ← {initNode}
5       closedSet ← {}
6       WHILE openSet IS NOT EMPTY
7           FOR EACH node IN openSet
8               node.Cost ← c(node.State) + h(node.State, g)
9           current ← openSet.GetNodeWithMinimumCost(g)
10          isGoalState ← ζ(g, current.Plan)
11          IF isGoalState
12              π ← node.ChainPlansBackward()
13              π̇ ← DeorderPlan(r0, π)
14              RETURN π̇
15          ELSE
16              openSet.Remove(current)
17              closedSet.Add(current)
18              FOR EACH agent IN cswAgents
19                  executablePlans ← agent.GenerateExecutablePartialOrderedPlans(current.State, g)
20                  FOR EACH π IN executablePlans
21                      r ← E(π, current.State)
22                      newNode ← [Plan: π, State: r, Cost: ∞, Parent: current]
23                      IF newNode NOT EXISTS IN openSet
24                          openSet.Add(newNode)
25      RETURN failure
```

Figure 3. Pseudo code of GenerateMultiAgentPlan algorithm.

The output of the cost function is a value between 0 and 2, and nodes with lower costs are more likely to lead to the final state. Having the costs, the node with minimum cost in the open set is considered as the current search node (lines 7 to 9, Figure 3). If the current node satisfies the goal condition, the plan

of that node is chained with the plans of its successive parents, a deordering algorithm (Figure 4) [54] is applied on the resulting plan, and the resulting plan is returned as the composition plan (lines 10 to 14, Figure 3). The deordering algorithm tries to move data retrieval operations which do not rely on the execution of the previous operations, to the initial steps of a partial-ordered plan.

```
1   DeorderPlan(r, π)
2       FOR k ← 2 TO P.size
3           FOR EACH a IN π[k]
4               FOR l ← 1 TO k
5                   previousState ← ExecutePlan(r, π, l − 1)
6                   isApplicableOnState ← ζ(condition(a), previousState)
7                   IF isApplicableOnState
8                       P.remove(a, k)
9                       P.insertInto(a, l)
10                      BREAK
11      FOR k ← 1 TO P.size
12          IF π[k].isEmpty
13              π.remove(k)
14      RETURN P
```

Figure 4. Pseudo code of the deordering algorithm.

If the current node does not satisfy the goal, the current node is supposed to be expanded by the CSW services. First, the current node is removed from the open set and added to the closed set. Then the partial-ordered plans, which are executable on the state of the current node *current.State*, are generated by each CSW agent through calling the *GenerateExecutablePartialOrderedPlans* method. Having the partial-ordered plans, the current node is expanded, and new search nodes are created and added to the open set (lines 16 to 24, Figure 3). To expand the current node using an executable partial-ordered plan, the received plan π, is applied on the state of the current node, *current.State*. The result state r, along with π are used to create a new search node, *newNode*, which is added to the open set afterwards (lines 21 to 24, Figure 3). This best node selection, condition checking, and expansion mechanism continuous till the goal state is reached or there is no search node for expansion in the open set. Figure 5 figuratively shows a simplified execution of the algorithm, where two CSW agents work with a composition coordinator to expand the search graph and find the composition plan.

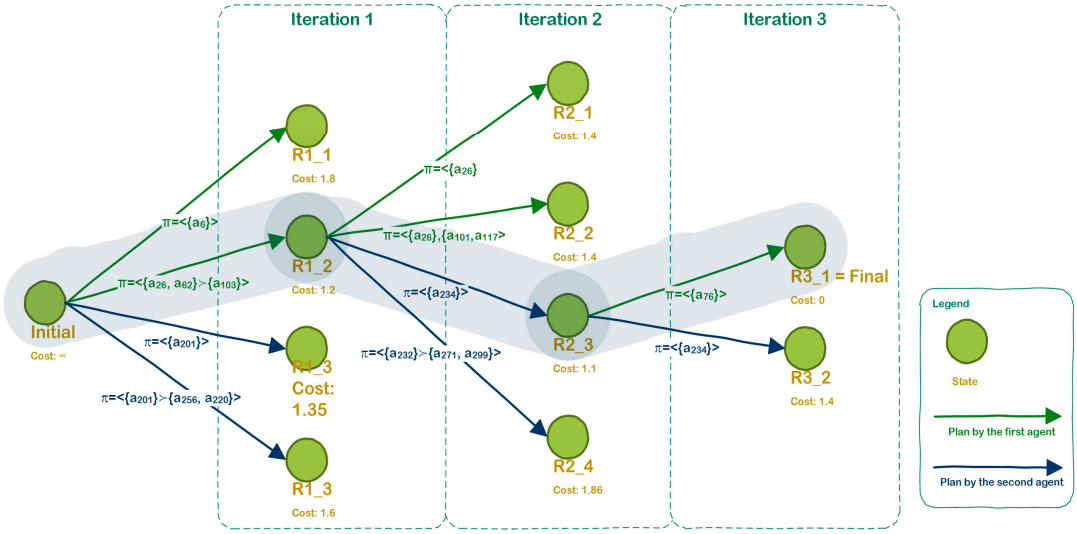

Figure 5. A simple example of the composition procedure.

3.4.2. CSW Component

To be able to work in the proposed multi-agent AI planning algorithm, a CSW service has to provide a *GenerateExecutablePartialOrderedPlans* method. The method receives a current state r_c, and returns a list of partial-ordered plans executable on r_c. The developed tree expansion algorithm

is represented in Figure 6. Starting from r_c, the algorithm creates an initial node; an open set (with initial node as its only member); a closed set; and a list of executable partial-ordered plans (lines 2 to 6, Figure 6). While the open set is not empty, the algorithm randomly selects a current node from the open set, removes the current node from the open set, and then adds it the closed set (lines 8 to 11). The operations that are applicable to the state of the current node and do not exist in the plan of the current node are selected afterwards (lines 12 to 16, Figure 6). Having a list of applicable operations, the current node is expanded by the applicable operations and the new nodes are added to the list of executable partial-ordered plans (lines 17 to 23, Figure 6). Finally, the algorithm returns the list of executable partial-ordered plans to the composition coordinator.

```
1   GenerateExecutablePartialOrderedPlans(r_c)
2       A ← GetRegisteredOWSOperators()
3       initNode ← {Plan: ⟨∅⟩, State: r_c, Cost: ∞, Parent: ∅}
4       openSet ← {initNode}
5       closedSet ← {}
6       executablePOPlans ← {}
7       WHILE openSet IS NOT EMPTY
8           current ← openSet.GetRandomNode()
9           openSet.remove(current)
10          closedSet.add(current)
11          applicableOperators ← {}
12          FOR EACH a IN A
13              isApplicableOnState ← ζ(a.condition, current.State)
14              isNotRepeated ← IsNotRepeatedInPlan(a, current.Plan)
15              IF isApplicableOnState AND isNotRepeated
16                  applicableOperators.add(a)
17          FOR EACH a IN applicableOperators
18              π ← ⟨{a}⟩
19              r ← ε(a, current.State)
20              child ← {Plan: π, State: r, Cost: ∞, Parent: current}
21              IF child NOT IN openSet AND child NOT IN closedSet
22                  openSet.add(child)
23                  executablePOPlans.add(child.ChainPlansBackward())
24      RETURN executablePOPlans
```

Figure 6. Pseudo code of *Generate Executable Partial Ordered Plans* method of Catalogue Service for Web (CSW).

4. Implementation

The proposed multi-agent AI planning algorithm for automatic composition of OWSs was developed and implemented in this study. The composition coordinator component and the CSW service have been written in Java programming language. The RDF handling functionalities along with the SPARQL 1.1 support have been provided through Apache Jena API. OWSs have been implemented in GeoServer.

4.1. Case study: Site Selection for Sheltering

To demonstrate the applicability of the proposed solution, the problem of site selection for sheltering after an earthquake based on spatial data and processes, which are available online through OWSs, is selected as the case study. It is assumed that participating disaster management organizations have published their spatial data and functionalities through OWSs in two separate CSW services: Regional CSW of Tehran and National CSW of Iran SDI. These CSW services are registered in the geoportal of the Emergency Management Organization (EOC) of Tehran. Table 1 shows each of the CSW services and the registered OWSs of the participating organizations.

Having access to the CSW services and the OWSs, a disaster management planner in the EOC of Tehran must be able to ask the composition coordinator agent to automatically generate a composite web service based on the atomic OWSs registered in the CSW services. The output web service must return polygon features with the following characteristics:

- The polygons can be vacant areas, parks, or forests.
- The area within the selected polygon features should be flat.
- The selected polygons should not be adjacent to damaged areas.

Table 1. CSW services along with registered OWSs.

CSW	Registered OGC Web Services			
	Web Service	**Organization**	**Type**	**Description**
Regional CSW of Tehran	wfsMun	Tehran Municipality	WFS	This service presents the municipal data. Layers: Parcel, Park, Vacant Land, etc.
	wfsEOC	Emergency Operation Center (EOC) of Tehran	WPS	This service returns a polygon of affected areas with dense building damage. Layers: Affected_Area
	wpsEOC_EvacuationPlanning	Emergency Operation Center (EOC) of Tehran	WFS	This service handles the site selection processes. In the case of sheltering, it receives various input datasets and calculates suitable evacuation sheltering sites.
National CSW of Iran SDI	wfsFRW	Forest and Watershed Management Organization	WFS	This service presents the spatial data from the Forest and Watershed management organization. Layers: Forest_Park, Watershed, Land Cover, etc.
	wcsNCC	National Cartographic Center	WCS	This service presents the elevation data for the entire country as coverage data. Layers: DEM
	wpsNCCSurface	National Cartographic Center	WPS	This service performs all of the surface operations, including slope, aspect, contour, hillshade, and viewshed generation.
	wpsNCCCT	National Cartographic Center	WPS	This service performs coordinate transformations. It provides two processes: ProjectFeatureDataset and ProjectCoverageDataset.

Considering the specified characteristics of the evacuation site, the disaster management planner sends their initial state and composition goal to the composition coordinator component. Figure 7 depicts the RDF graph of the initial state, which defines the bounding box of the study area and required spatial reference system. The SPARQL ask query that defines the user goal by expressing the characteristics of the desired output sites is shown in Figure 8. Figures 9 and 10 represent the condition and effect of project feature operation of wpsNCCCT WPS (Web Processing Service) service (Table 1), respectively.

```
<?xml version="1.0"?>
<rdf:RDF xml:base=".../initial" xmlns=".../initial#" xmlns:agt=".../Agent#" xmlns:owl=".../owl#" xmlns:rdf=".../22-rdf-syntax-ns#"
xmlns:rdfs=".../rdf-schema#" xmlns:gms=".../GMS#" xmlns:xsd=".../XMLSchema#">
    <rdf:Description rdf:about=".../initial#inputEnvelope">
        <gms:upperCorner>35.8080 51.5529</gms:upperCorner>
        <gms:srsName rdf:datatype=".../XMLSchema#anyURI">ogc:def:crs:EPSG:4326</gms:srsName>
        <gms:lowerCorner>35.6100 51.2846</gms:lowerCorner>
        <rdf:type>
            <rdf:Description rdf:about="gms:Envelope"/>
        </rdf:type>
    </rdf:Description>
    <rdf:Description rdf:about="#user">
        <rdf:type>
            <rdf:Description rdf:about=".../Agent#WebServiceUserAgent"/>
        </rdf:type>
        <agt:hasKnowledgeAbout>
            <rdf:Description rdf:about="#inputEnvelope"/>
        </agt:hasKnowledgeAbout>
    </rdf:Description>
</rdf:RDF>
```

Figure 7. Initial state for composing OWSs

```
# Goal
PREFIX rdf:<.../22-rdf-syntax-ns#>
PREFIX agt:<.../Agent#>
PREFIX cso:<.../COSMO.owl>
PREFIX gms:<.../GMS#>
PREFIX csoext:<.../COSMO-Ext#>
ASK
WHERE
{
        ?u rdf:type <.../Agent#WebServiceUserAgent>.
        ?u agt:hasKnowledgeAbout ?featDS.
        ?featDS <.../COSMO.owl#hasAttribute> <csoext:SuitableForEvacuationInEarthquakeEvent>.
        ?featDS rdf:type <cso:Place>.
        ?featDS gms:typeName ?tn.
        ?featDS gms:boundedBy ?env.
        ?env gms:srsName ?srs.
        ?env gms:lowerCorner ?lowCorner.
        ?env gms:upperCorner ?uprCorner.
        FILTER (regex(str(?srs), "ogc:def:crs:EPSG:4326") &&
                regex(str(?lowCorner), "35.6100 51.2846") &&
                regex(str(?uprCorner), "35.8080 51.5529")).
}
```

Figure 8. User goal for composing OWSs.

```
# Condition of NCC_CT_WPS Execute Project Feature
PREFIX rdf:<.../22-rdf-syntax-ns#>
PREFIX agt:<.../Framework/Agent#>
PREFIX wps:<.../Framework/WPS#>
PREFIX ctwps:<.../NCC_CT_WPS#>
PREFIX gmc:<.../GMC#>
PREFIX gms:<.../GMS#>
PREFIX xsd:<.../XMLSchema#>
ASK
WHERE
{
        ?u rdf:type <.../Agent#WebServiceUserAgent>.
        ?u agt:hasKnowledgeAbout ?projectFeatureDataset.
        ?projectFeatureDataset rdf:type <wps:Process>.
        ?projectFeatureDataset wps:processIdentifier ?pProjectFDSId.
        ?projectFeatureDataset wps:hasInput ?processInFeatDS.
        ?processInFeatDS rdf:type <gms:FeatureCollection>.
        ?projectFeatureDataset wps:hasInput ?processOutSrsName.
        ?processOutSrsName rdf:type <xsd:integer>.
        ?projectFeatureDataset wps:hasOutput ?processOutFeatDS.
        ?processOutFeatDS rdf:type <gms:FeatureCollection>.
        ?u agt:hasKnowledgeAbout ?inFeatDS.
        ?inFeatDS rdf:type ?inFeatDsRdfType.
        ?inFeatDS rdf:type <gms:FeatureCollection>.
        ?inFeatDS gms:typeName ?inFeatDsTypeName.
        ?inFeatDS gms:geomProperty ?inFeatDsGeomProperty.
        ?inFeatDS gms:boundedBy ?inFeatDSBBox.
        ?inFeatDSBBox gms:srsName ?inFeatDSBBoxSrsName.
        ?u agt:hasKnowledgeAbout ?inputEnv.
        ?inputEnv rdf:type <gms:Envelope>.
        ?inputEnv gms:srsName ?outSrsName.
        FILTER (regex(str(?pProjectFDSId), "geopack:ProjectFeatureDataset") &&
                        (str(?inFeatDSBBoxSrsName) != str(?outSrsName))
}
```

Figure 9. Condition of NCC_CT_WPS Execute Project Feature operation.

```
# Effect of NCC_CT_WPS Execute Project Feature
PREFIX rdf:<.../22-rdf-syntax-ns#>
PREFIX agt:<.../Agent#>
PREFIX wps:<.../WPS#>
PREFIX ctwps:<.../NCC_CT_WPS#>
PREFIX gmc:<.../GMC#>
PREFIX gms:<.../GMS#>
PREFIX xsd:<.../XMLSchema#>
DELETE
{
        ?inFeatDSBBox gms:srsName ?inFeatDSBBoxSrsName.
}
INSERT
{
        ?u agt:hasKnowledgeAbout _:outFeatDS.
        _:outFeatDS agt:isGeneratedBy <ctwps:NCC_CT_WPS_Execute_ProjectFeatureDataset>.
        _:outFeatDS rdf:type ?inFeatDsRdfType.
        _:outFeatDS rdf:type <gms:FeatureCollection>.
        _:outFeatDS dc:title ?inFeatDsTitle.
        _:outFeatDS dc:abstract ?inFeatDsAbstract.
        _:outFeatDS gms:typeName ?inFeatDsTypeName.
        _:outFeatDS gms:geomProperty ?inFeatDsGeomProperty.
        _:outFeatDS gms:boundedBy ?inFeatDSBBox.
        ?inFeatDSBBox gms:srsName ?outSrsName.
}
WHERE
{
        # Defined based on the condition of the operation
}
```

Figure 10. Effect of NCC_CT_WPS Execute Project Feature operation.

4.2. Planning Results

The composition coordinator component and both CSW services were implemented in a test environment (Figure 11 shows the initial settings of the execution environment). Providing the initial state and the composition goal of the case study as input arguments, the GenerateMultiAgentPlan method of the composition coordinator component was called.

```
================================================================================
  Multi Agent Planning
================================================================================
--------------------------------------------------------------------------------
  Initial settings:
--------------------------------------------------------------------------------
  Coordinator Agent: 1092208918
       Registered CSWs:
       1: Regional CSW of Tehran
              OGC Web Service List:
                     wfsMun
                     wfsEoc
                     wpsEocEvacuationPlanning
       2: CSW of Iran National SDI
              OGC Web Service List:
                     wfsFrw
                     wpsNccCt
                     wpsNccSurface
                     wcsNcc
--------------------------------------------------------------------------------
```

Figure 11. Initial settings of the execution environment.

The composition coordinator was able to find a valid plan after interaction with CSW agents. Figure 12 shows the resultant plan.

```
--------------------------------------------------------------------------------
  Result
--------------------------------------------------------------------------------
      Generated Plan:
            Step 1
                  wfsMun.GetCapabilities
                  wfsEoc.GetCapabilities
                  wfsFrw.GetCapabilities
                  wcsNcc.GetCapabilities
                  wpsNccSurface.GetCapabilities
                  wpsNccCt.GetCapabilities
                  wpsEocEvacuationPlanning.GetCapabilities
            Step 2
                  wfsMun.DescribeFeatureType
                  wfsEoc.DescribeFeatureType
                  wfsFrw.DescribeFeatureType
                  wcsNcc.DescribeCoverage
                  wpsNccSurface.DescribeProcess
                  wpsNccCt.DescribeProcess
                  wpsEocEvacuationPlanning.DescribeProcess
            Step 3
                  wfsMun.GetFeature
                  wfsEoc.GetFeature
                  wfsFrw.GetFeature
                  wcsNcc.GetCoverage
            Step 4
                  wpsNccSurface.Execute_Slope
                  wpsNccCt.Execute_ProjectFeatureDataset
            Step 5
                  wpsEocEvacuationPlanning.Execute_EarthquakeEvacuationSite
--------------------------------------------------------------------------------
            NumberOfOperations: 21
================================================================================
```

Figure 12. Result plan.

4.3. Translation to BPEL and Execution

The output plan was translated to an XML-based orchestration language called BPEL [55]. Using BPEL, the composite web service can be described and then an orchestration engine will able to execute the composite web service. In this study, the output plan was automatically translated into a BPEL process document. The BPEL document was registered in the Java Business Integration (JBI) runtime over the GlassFish server. The JBI runtime environment includes the BPEL Service Engine, which provides functionalities for executing BPEL processes. The JBI environment presents the composite OWS as a web service. Providing the required inputs, the web service was executed and returned appropriate sites for sheltering.

4.4. Performance Demonstration

Since the number of available OWSs in the case study was limited and there was no testbed of semantically described OWSs in the geomatics community, a service simulator program was developed that could produce computer-generated OWSs. Using the simulated OWSs the feasibility of the proposed multi-agent planning algorithm in dealing with real-world, complex circumstances was tested. The goal of the test was to demonstrate how adding new CSW services to the geoportal could increase the performance of the composition generation process.

The service simulator program was developed so that it can generate OWSs, and then semantically annotate them based on predefined templates. A list of ninety-two tuples, each including the required information for generation of a simulated OWS was compiled. Each tuple contained a combination of information, including service type from OWS classification ontology, feature type from Geography Markup Language (GML) [56] ontology or GML-coverage profile ontology, class name from the fundamental ontology or the domain ontology, spatial reference identifier, and a random bounding box.

The service simulator application looped through the list and by replacing the required information of each tuple, generated ninety-two OWSs.

The test process utilized the ninety-two simulated OWSs along with the eight OWSs of the case study as the testbed. A network of six virtual nodes with Ubuntu 16.04.3 operating systems on Oracle VirtualBox virtualization software, where each node had two gigabytes of RAM and two CPUs, were used so that on the first node the composition coordinator component was installed and on the five other nodes, five CSW services were deployed. The test started with the composition coordinator node and two CSW service nodes, where each CSW service had fifty OWSs registered in it. The initial state and the goal state of the case study were sent to the composition coordinator component and the execution time was saved. In the next step, another CSW service node was added, the one hundred OWSs were registered in the three CSW services and the execution time was saved. This process continued to the point all five CSW service nodes were used.

To account for unwanted effects from the background processing of the operating systems and the virtualization software, the test process was run 20 times. Figure 13 presents the execution results of the different settings of the CSW services for 20 iterations. For the combination of one coordinator node, two CSW services, and one hundred registered OWSs, on average, the system was able to solve the composition problem in 20 s, which is a reasonable time for a web services composition task. As the number of CSW nodes increased, the composition time of the system of one hundred OWSs reduced exponentially, so that for five CSW services the average execution time was 0.3 s. This reduction proved that by using the computational power of CSW services, the proposed distributed OWS composition solution horizontally scaled up by adding new nodes. Therefore, it is less vulnerable to the exponential growth of the search space and would be able to deal with complex problems of the real world.

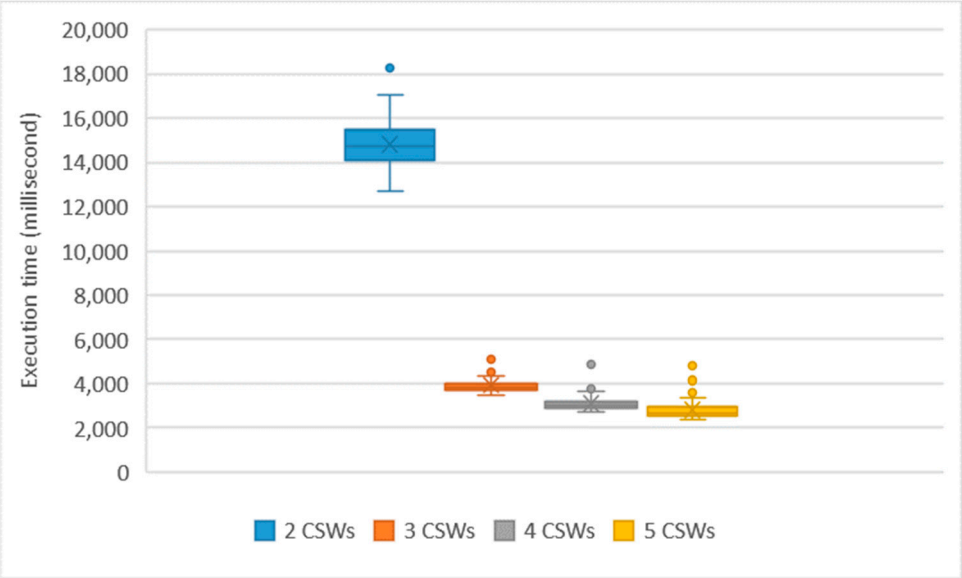

Figure 13. Execution time for different number of CSW nodes.

5. Discussion

The proposed solution in this study was able to address the two critical challenges of the automatic composition of OWSs in geoportals mentioned in the introductory section. The algorithm is distributed and compatible with the distributed architecture of geoportals. It does not need to fetch the information about all registered OWSs in the CSW nodes to a central node and generate a vast search space. Instead, the computational powers of CSW nodes are exploited, so that with the help of a coordinator component, CSW nodes can collaboratively solve the composition problem. While the case-study showed that the system could solve the automatic composition problem in a real-world scenario, the performance demonstration in a simulation environment proved that by adding new CSW nodes

to the geoportal, the performance of the algorithm increased exponentially, which meant that the system was horizontally scalable.

To apply the proposed solution in any geoportal, we need to add a composition coordinator component that implements the automatic composition algorithm (Section 3.4.1), and also force the CSW services that are registered in the geoportal to implement the *Generate Executable Partial Ordered Plans* method and its underlying algorithm (Section 3.4.2), in addition to the standard methods of CSW. This additional method of CSW services can be considered as a possible extension to the CSW standard in future revisions or simply be provided through a standard WPS [46] interface. In this condition, the geoportal will be able to automatically compose the semantically annotated OWSs using the proposed solution. Meanwhile, a motivating resolution for widespread adoption of the proposed solution is to extend opensource geoportal software, e.g., GeoNetwork (https://geonetwork-opensource.org/), GeoNode (http://geonode.org/), pyCSW (http://pycsw.org/) and Deegree (https://www.deegree.org/), and add the required functionalities to them, so that they can be easily implemented in spatial data infrastructures. Among available opensource geoportal software, GeoNetwork seems to be more appropriate to be extended because it already has broad supports for working with SPARQL and RDF.

In the proposed solution, geospatial aspects of OWSs are semantically annotated using GML and GML-coverage profile ontology, and SPARQL is used as the query language. Meanwhile, GeoSPARQL [57] has been developed and commonly accepted in the geospatial community for handling semantic geospatial data. GeoSPARQL provides a framework for the semantic description of geospatial features based on GML and OGC Simple Feature [58] ontologies, along with a SPARQL query interface. With these in mind, a question may arise as to why GeoSPARQL were not used in this study? The proposed solution of this study works based on the update functionalities of SPARQL 1.1 Update Standard (see https://www.w3.org/TR/sparql11-update/ and Section 4.3 of the paper). At the time of writing this article, there was no semantic web engine that supported both GeoSPARQL and SPARQL 1.1 Update standard. Therefore, GeoSPARQL was not used for the definition of the conditions and effects of OWSs in the case study. However, as soon as the leading semantic web engines, e.g., Apache Jena, support GeoSPARQL, the developed system can be modified to work with GeoSPARQL. This is just a discussion about possible tools and does not influence the approach used in this study.

To evaluate the performance of a web service composition solution, researchers typically use a testbed of sample web services (see References [59–61]). However, in the geospatial community, there is no testbed of semantically annotated OWSs to be used for evaluation of the proposed composition methods. Owing to this limitation, other researchers, e.g., Yue et al. [5], Cruz, Monteiro, and Santos [6], Du et al. [32], Al-Areqi, Lamprecht, and Margaria [9], have proved the feasibility of their OWS composition solution using case studies with limited numbers of OWSs. However, there are still issues related to the test of the performance. To address these issues, in this study, a service simulator program was developed to simulate a testbed of 100 OWSs. The simulated services on six computational nodes were utilized to test the ability of the algorithm to deal with real-world circumstances. However, generation of a standard testbed of semantically annotated OWSs with thousands of OWSs seems to be necessary to be considered by the geospatial community. Such testbeds will provide the ability to benchmark and compare the performances of proposed OWS composition solutions in future.

Based on OGC Catalog Service Standard [36], CSW can be cascaded by other CSW services. Additionally, an OWS may be published through multiple CSW services. The complexity resulting from multiple access points to an OWS service and repetition of OWSs in CSW services, should be addressed in the future extension of the proposed solution. Utilization of a URI (Unified Resource Identifier) as a global identifier of OWSs as proposed in the linked data principals [62] can be an answer in this regard. Including the global identifier of OWSs in the partial-ordered plans that are communicated between the composition coordinator component and the CSW nodes, will help the

CSW nodes to distinguish the same OWSs, and therefore avoid using OWSs that have already been added to the composition by other nodes.

The proposed algorithm in this study, was developed based on A* algorithm. Using a heuristic function to lead the search protects the algorithm from the explosion of the search space and expansion of the search tree in arbitrary directions. Using an open set containing all search nodes that have not yet been explored ensures the completeness of the algorithm. Nissim and Brafman [40] proved that a similar distributed A* algorithm was both complete and optimal.

The proposed algorithm does not use semantic similarity measures and semantic matching techniques to detect similar concepts, which have been defined with different terminologies. Addressing the requirement for similarity recognition in the composition process is an important issue that should be considered as another future work. Additionally, considering data quality and quality of service in the multi-agent AI planning algorithm is a future work for this research. This approach should also be implemented on a large-scale geoportal so that its pros and cons can be thoroughly evaluated.

6. Conclusions

This article presents a solution for automatic composition of OWSs. A multi-agent AI planning algorithm has been introduced which exploits the computational power of CSW services to generate composite web services out of atomic OWSs. The proposed solution delegates the processing overload of the web service composition to the CSW service nodes. The solution horizontally scales up by registering new CSW service nodes. Therefore, it is less vulnerable to the exponential growth of the search space of the AI planning algorithms. It is also an extension to the distributed architecture of geoportals, which has been widely exploited in the geospatial community, and is therefore an appropriate choice to be implemented and used in real-world scenarios.

To demonstrate the functionality of the proposed solution, a prototype system was developed. The system was used to solve geospatial web service composition problems in a case-study related to site selection for sheltering after an earthquake. In a collaboration among the components of the geoportal, including the CSW services, the geoportal was able to automatically generate the requested composition out of the OWSs which were registered in the CSW services. The performance of the solution was also tested in a simulation environment with one hundred OWSs and five CSW services. The test showed that the solution was fast and horizontally scalable, so that by adding new CSW services, the execution time reduced exponentially.

Implementation of the proposed solution in futuristic spatial data infrastructures and resource publishing environments, e.g., the European INSPIRE Geoportal or the NSDI Clearinghouse of the United States, can facilitate the process of dissemination, integration, and utilization of spatial data and functionality from various sources, which eventually leads to providing value-added services for users. Additionally, the proposed multi-agent algorithm can be used in cloud computing environments as a broker that helps consumers to compose the data and functionalities out of the pool of services provided by service providers.

Author Contributions: M.F. participated in the conceptualization of the study, proposing the method, development of the prototype, testing and writing the text. A.M. contributed to the research design, discussing the results, reviewing and revising the text.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yue, P.; Di, L.; Yang, W.; Yu, G.; Zhao, P. Semantics-based automatic composition of geospatial Web service chains. *Comput. Geosci.* **2007**, *33*, 649–665. [CrossRef]

2. Klusch, M. Semantic web service coordination. In *CASCOM: Intelligent Service Coordination in the Semantic Web*; Schumacher, M., Schuldt, H., Helin, H., Eds.; Birkhäuser: Basel, Switzerland, 2008; pp. 59–104.
3. Bartalos, P.; Bieliková, M. Fast and Scalable Semantic Web Service Composition Approach Considering Complex Pre/Postconditions. In Proceedings of the 2009 IEEE Congress on Services, International Workshop on Web Service Composition and Adaptation, Los Angeles, CA, USA, 6–10 July 2009.
4. Kumar, S. A Multi-Agent Negotiation-Based Approach to Selection and Composition of Semantic Web Services. In *Agent-Based Semantic Web Service Composition, SpringerBriefs in Electrical and Computer Engineering*; Springer: New York, NY, USA, 2012; pp. 37–56.
5. Yue, P.; Di, L.; Yang, W.; Yu, G.; Zhao, P.; Gong, J. Semantic Web Services-based process planning for earth science applications. *Int. J. Geogr. Inf. Sci.* **2009**, *23*, 1139–1163. [CrossRef]
6. Cruz, S.A.B.; Monteiro, A.M.V.; Santos, R. Automated geospatial Web Services composition based on geodata quality requirements. *Comput. Geosci.* **2012**, *47*, 60–74. [CrossRef]
7. Farnaghi, M.; Mansourian, A. Automatic composition of WSMO based geospatial semantic web services using artificial intelligence planning. *J. Spat. Sci.* **2013**, *58*, 235–250. [CrossRef]
8. Farnaghi, M.; Mansourian, A. Disaster planning using automated composition of semantic OGC web services: A case study in sheltering. *Comput. Environ. Urban Syst.* **2013**, *41*, 204–218. [CrossRef]
9. Al-Areqi, S.; Lamprecht, A.-L.; Margaria, T. Constraints-Driven Automatic Geospatial Service Composition: Workflows for the Analysis of Sea-Level Rise Impacts. In *Computational Science and Its Applications—ICCSA 2016: 16th International Conference, Beijing, China, 4–7, July 2016, Proceedings, Part III*; Gervasi, O., Ed.; Springer International Publishing: Cham, Switzerland, 2016; pp. 134–150.
10. Ghallab, M.; Nau, D.S.; Traverso, P. *Automated Planning: Theory and Practice*; Elsevier: Amsterdam, The Netherlands; Morgan Kaufmann: Boston, FL, USA, 2004.
11. Nebert, D.D. *Developing Spatial Data Infrastructures: The SDI Cookbook—Version 2.0*; Global Spatial Data Infrastructure (GSDI): Paris, France, 2004.
12. Nebert, D.; Whiteside, A.; Vretanos, P.A. OpenGIS Catalogue Services Specification, Version 2.0.2. *OGC 07-006r1*. 2007. Retrieved 25 January 2013. Available online: http://www.opengeospatial.org/standards/cat (accessed on 6 August 2018).
13. Baryannis, G.; Plexousakis, D. *Automated Web Service Composition: State of the Art and Research Challenges*; Tech. Rep. 409; ICS-FORTH: Heraklion, Greece, October 2010.
14. El Falou, M.; Bouzid, M.; Mouaddib, A.-I.; Vidal, T. A complete and optimal distributed algorithm based on global heuristic for Web services composition. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Istanbul, Turkey, 10–13 October 2010.
15. El Falou, M.; Bouzid, M.; Mouaddib, A.-I.; Vidal, T. A distributed multi-agent planning approach for automated web services composition. *Web Intell. Agent Syst.* **2012**, *10*, 423–445. [CrossRef]
16. Lin, S.-Y.; Lin, G.-T.; Chao, K.-M.; Lo, C.-C. A Cost-Effective Planning Graph Approach for Large-Scale Web Service Composition. *Math. Probl. Eng.* **2012**, *2012*, 783476. [CrossRef]
17. Maguire, D.J.; Longley, P.A. The emergence of geoportals and their role in spatial data infrastructures. *Comput. Environ. Urban Syst.* **2005**, *29*, 3–14. [CrossRef]
18. Bernard, L.; Kanellopoulos, I.; Annoni, A.; Smits, P. The European geoportal—-one step towards the establishment of a European Spatial Data Infrastructure. *Comput. Environ. Urban Syst.* **2005**, *29*, 15–31. [CrossRef]
19. Beaumont, P.; Longley, P.A.; Maguire, D.J. Geographic information portals—A UK perspective. *Comput. Environ. Urban Syst.* **2005**, *29*, 49–69. [CrossRef]
20. Alameh, N. Service chaining of interoperable geographic information web services. *IEEE Internet Comput.* **2002**, *7*, 22–29. [CrossRef]
21. Lemmens, R.; De By, R.; Gould, M.; Wytzisk, A.; Granell, C.; Van Oosterom, P. Enhancing Geo-Service Chaining through Deep Service Descriptions. *Trans. GIS* **2007**, *11*, 849–871. [CrossRef]
22. Lutz, M.; Lucchi, R.; Friis-Christensen, A.; Ostländer, N. A Rule-Based Description Framework for the Composition of Geographic Information Services. In *GeoSpatial Semantics*; Fonseca, F., Rodríguez, M., Levashkin, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4853, pp. 114–127.
23. Di, L.; Yue, P.; Yang, W.; Yu, G.; Zhao, P.; Wei, Y. Ontology-Supported Automatic Service Chaining for Geospatial Knowledge Discovery. In Proceedings of the American Society of Photogrammetry and Remote Sensing, Tampa, FL, USA, 7–11 May 2007.

24. Gone, M.; Schade, S. Towards Semantic Composition of Geospatial Web Services–Using WSMO instead of BPEL. *Int. J. Spat. Data Infrastruct. Res.* **2008**, *3*, 192–214.
25. Friis-Christensen, A.; Lucchi, R.; Lutz, M.; Ostländer, N. Service Chaining Architectures for Applications Implementing Distributed Geographic Information Processing. *Int. J. Geogr. Inf. Sci.* **2009**, *23*, 561–580. [CrossRef]
26. Supavetch, S.; Chunithipaisan, S. Interface Independent Geospatial Services Orchestration. *Inf. Technol. J.* **2011**, *10*, 1126–1137. [CrossRef]
27. Hofer, B.; Mäs, S.; Brauner, J.; Bernard, L. Towards a knowledge base to support geoprocessing workflow development. *Int. J. Geogr. Inf. Sci.* **2017**, *31*, 694–716. [CrossRef]
28. Wiemann, S. Formalization and web-based implementation of spatial data fusion. *Comput. Geosci.* **2017**, *99* (Suppl. C), 107–115. [CrossRef]
29. Sheshagiri, M.; des Jardins, M.; Finin, T. A Planner for Composing Services Described in DAML-S. In Proceedings of the ICAPS'03 Workshop on Planning for Web Services, Trento, Italy, 10 June 2003.
30. Roman, D.; Keller, U.; Lausen, H.; de Bruijn, J.; Lara, R.; Stollberg, M.; Polleres, A.; Feier, C.; Bussler, C.; Fensel, D. Web service modeling ontology. *Appl. Ontol.* **2005**, *1*, 77–106.
31. Vitvar, T.; Kopecky, J.; Viskova, J.; Fensel, D. Wsmo-Lite Annotations for Web Services. In Proceedings of the 5th Annual European Semantic Web Conference (ESWC 2008), Tenerife, Spain, 1–5 June 2008.
32. Du, W.; Fan, H.; Li, J.; Wang, H. Model-driven geospatial web service composition. In Proceedings of the ISPRS Technical Commission VI Symposium, Wuhan, China, 19–21 May 2014; Volume XL-6.
33. Vaithiyanathan, R.; Govindharajan, T.A. User preference-based automatic orchestration of web services using a multi-agent. *Comput. Electr. Eng.* **2015**, *45*, 68–76. [CrossRef]
34. Küngas, P.; Matskin, M. Semantic Web Service Composition Through a P2P-Based Multi-agent Environment. In *Agents and Peer-to-Peer Computing*; Despotovic, Z., Joseph, S., Sartori, C., Eds.; Springer: Berlin/Heidelberg, Germay, 2006; Volume 4118, pp. 106–119.
35. Charif, Y.; Sabouret, N. Dynamic service composition enabled by introspective agent coordination. *Auton. Agents Multi-Agent Syst.* **2013**, *26*, 54–85. [CrossRef]
36. Nebert, D.; Voges, U.; Bigagli, L. OGC Catalogue Services 3.0—General Model. *OGC 12-168r6*. 2016. Retrieved 1 May 2018. Available online: http://docs.opengeospatial.org/is/12-168r6/12-168r6.html (accessed on 6 August 2018).
37. Mansourian, A.; Omidi, E.; Toomanian, A.; Harrie, L. Expert system to enhance the functionality of clearinghouse services. *Comput. Environ. Urban Syst.* **2011**, *35*, 159–172. [CrossRef]
38. Jennings, N.R. Coordination techniques for distributed artificial intelligence. In *Foundations of Distributed Artificial Intelligence*; O'Hare, G., Jennings, N.R., Eds.; John Wiley & Sons: New York, NY, USA, 1996; pp. 187–210.
39. Weerdt, M.D.; Mors, A.T.; Witteveen, C. Multi-agent planning: An introduction to planning and coordination. In Proceedings of the European Agent Systems Summer School, Barcelona, Spain, 6–10 July 2005.
40. Nissim, R.; Brafman, R.I. Multi-agent A* for parallel and distributed systems. In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, Valencia, Spain, 4–8 June 2012; Volume 3.
41. Torreño, A.; Onaindia, E.; Komenda, A.; Štolba, M. Cooperative Multi-Agent Planning: A Survey. *ACM Comput. Surv. CSUR* **2017**, *50*, 84. [CrossRef]
42. Sirin, E. Automated Composition of Web Services Using AI Planning Techniques. Master's Thesis, University of Maryland, College Park, MD, USA, 2004.
43. Beaujardiere, J.D.L. Web Map Service, Version 1.3. *OGC 04-024*. 2004. Retrieved 10 October 2012. Available online: http://www.opengeospatial.org/standards/wms (accessed on 6 August 2018).
44. Vretanos, P.A. Web Feature Service Implementation Specification, Version 1.1.0. *OGC 04-094*. 2005. Retrieved 25 October 2012. Available online: http://www.opengeospatial.org/standards/wfs (accessed on 6 August 2018).
45. Evans, J.D. Web Coverage Service (WCS). Version 1.0.0. *OGC 03-065r6*. 2003. Retrieved 10 October 2012. Available online: http://www.opengeospatial.org/standards/wcs (accessed on 6 August 2018).
46. Schut, P. OpenGIS Web Processing Service, Version 1.0.0. *OGC 05-007r7*. 2007. Retrieved 27 January 2013. Available online: http://www.opengeospatial.org/standards/wps (accessed on 6 August 2018).

47. Babitski, G.; Bergweiler, S.; Hoffmann, J.; Schön, D.; Stasch, C.; Walkowski, A. Ontology-based Integration of Sensor Web Services in Disaster Management. In *GeoSpatial Semantics*; Janowicz, K., Raubal, M., Levashkin, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5892, pp. 103–121.
48. Kopecky, J.; Vitvar, T.; Bournez, C.; Farrell, J. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Comput.* **2007**, *11*, 60–67. [CrossRef]
49. Prud'hommeaux, E.; Seaborne, A. SPARQL Query Language for RDF. *W3C Recommendation*. 2008. Retrieved 9 October 2012. Available online: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/ (accessed on 6 August 2018).
50. Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T. OWL-S: Semantic markup for web services. *W3C Member Submission*. 2004. Retrieved 15 October 2012. Available online: http://www.w3.org/Submission/OWL-S/ (accessed on 6 August 2018).
51. Fensel, D.; Bussler, C. The Web Service Modeling Framework WSMF. *Electron. Commer. Res. Appl.* **2002**, *1*, 113–137. [CrossRef]
52. Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*, 3rd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2010.
53. Whitesitt, J.E. *Boolean Algebra and Its Applications*; Courier Corporation: North Chelmsford, MA, USA, 2012.
54. Backstrom, C. Computational Aspects of Reordering Plans. *J. Artif. Intell. Res.* **1998**, *9*, 99–137. [CrossRef]
55. Alves, A.; Arkin, A.; Askary, S.; Barreto, C.; Bloch, B.; Curbera, F.; Ford, M.; Goland, Y.; Guízar, A.; Kartha, N.; et al. Web Services Business Process Execution Language Version 2.0. *OASIS Standard*. 2007. Retrieved 10 October 2012. Available online: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (accessed on 6 August 2018).
56. Brink, L.V.D.; Portele, C.; Vretanos, P.A. Geography Markup Language (GML) Simple Features Profile. 2010. Retrieved 15 July 2012. Available online: http://www.opengeospatial.org/standards/gml (accessed on 6 August 2018).
57. Perry, M.; Herring, J. OGC GeoSPARQL—A Geographic Query Language for RDF Data. *OGC 11-052r4, OGC Implementation Standard*. 2012. Retrieved 25 January 2018. Available online: http://www.opengeospatial.org/standards/geosparql (accessed on 6 August 2018).
58. Herring, J.R. OpenGIS®Implementation Standard for Geographic Information. *OGC 06-103r4, OpenGIS Implementation Standard*. 2011. Retrieved 25 January 2018. Available online: http://www.opengeospatial.org/standards/sfa (accessed on 6 August 2018).
59. Yeganeh, S.H.; Habibi, J.; Rostami, H.; Abolhassani, H. Semantic web service composition testbed. *Comput. Electr. Eng.* **2008**, *36*, 805–817. [CrossRef]
60. SemWebCentral. OWLS-TC: OWL-S Service Retrieval Test Collection. Latest Version OWLSTC 4.0 (OWLS-TC4) Published on 21 September 2010. First Version of OWLS-TC Was Created by Benedikt Fries, Mahboob Khalid, Matthias Klusch (DFKI) and Published at Semwebcentral on 11 April 2005. Available online: http://projects.semwebcentral.org/projects/owls-tc/ (accessed on 6 August 2018).
61. Cabral, L.; Li, N. Building the WSMO-Lite Test Collection on the SEALS Platform. In Proceedings of the 9th Extended Sematic Web Conference 2012 (ESWC 2012), Heraklion, Greece, 27–31 May 2012; Volume 843.
62. Bizer, C.; Heath, T.; Berners-Lee, T. Linked data-the story so far. *Int. J. Semant. Web Inf. Syst.* **2009**, *5*, 1–22. [CrossRef]