Information Modelling and Knowledge Bases XXXVI
Y. Kiyoki et al. (Eds.)

doi:10.3233/FAIA241589

Using MAS for a Sketch Map Creation

Marek MENŠÍK, Matěj TOMŠŮ, Petr RAPANT, Adam ALBERT,
VSB - Technical University of Ostrava, 17. listopadu 15, 708 00 Ostrava, Czech Republic

Abstract. Knowledge about the real world is often recorded in plain text, such as posts on social networks, descriptions in various guides, etc. These messages include spatial information that can be extracted using natural language processing methods. The extracted information can then be represented as a planar graph, which can be further transformed into a topological map using additional information describing the area. This paper outlines an algorithm that takes a given planar graph as input and uses a multi-agent system to place individual points in 2D space, creating a topological map respecting all edge directions given in the narratives.

Keywords. Spatial data, Planar graph, Sketch map, Map topology

1. Introduction

In this paper, we build upon [1], where we outlined the algorithm for creating a topological map by placing points representing locations obtained from narratives into a 2D space. Narratives are recorded in natural language, and it is necessary to transform them into a formal language for better machine processing.[1] The process of transforming from natural language to the formal language using TIL constructions is outlined in [2], and it is not the aim of this paper.

The process of creating a sketch map from plain text data consists of three steps: (i) identification of spatial entities (named entity recognition) and their spatial relations by natural language processing, (ii) creation of a planar graph that captures identified spatial entities and their spatial relations, and (iii) conversion of this graph into a topological graph and topological map. This paper presents the first results related to the third step, which was realized using the multi-agent approach.

The entire paper is structured as follows: In the following Chapter 2, a general strategy for the utilization of multi-agent systems (MAS) is described.[2] In Chapter 3, We describe the system designed by us, including its components and the rules based on which individual agents act and communicate; and in Chapter 4, we showcase the system's functionality using a specific example.

[1]See more in [2], [3].
[2]More about MAS can bee seen in [4].

2. MAS Strategy

MASs is a field of artificial intelligence and computer science study focusing on coordinating autonomous agents to achieve a common goal or solve complex problems through interaction and cooperation. The research in this domain spans various aspects, including consensus protocols, adaptive control, event-triggered mechanisms, communication strategies, and fault tolerance, among others.

MASs are applied across diverse fields, showcasing their versatility and effectiveness in addressing complex, dynamic, and distributed problems. Notable examples include the management of mobile robots and smart grids [5], as well as the optimization of oilfield pipeline networks [6]. In healthcare and medical systems, MASs facilitate object detection/tracking, control/assistance, security systems, and human-care systems, thereby improving service delivery and operational efficiency [7]. In process industries, such as wastewater treatment, MASs are utilized for process monitoring, helping to manage and optimize treatment processes to ensure environmental compliance and operational efficiency [8].

The deployment of agents can be addressed in several ways. One option is to design an algorithm for computing the positions of individual agents, and the other is to leave this computation to the individual agents within the MAS, solving the problem in a distributed manner. From our perspective, the latter solution is more robust, as it deals only with the behavior of individual agents, and a topological graph emerges emergently by placing individual agents in positions that respect the absolute directions.

The general principle of operation of a multi-agent system for creating a topological graph can be described as follows:

Start conditions:

- Each agent knows its neighbors and the absolute directions of its connections with these agents. The absolute directions and their numbering are described below.
- For the optimization of topological graph creation, it is advantageous when agents are inserted into the system along individual bounding circles (Definition 1). This way, each agent has exactly two neighbors.

System behaviour:

1. The first agent is placed in the MAS and assigned an absolute position at the origin of the coordinate system ([0,0]).
2. Another agent is placed in the MAS space.
   - 2.1 If none of the neighboring agents is present in the MAS space, the agent is placed at the origin of the coordinate system ([0,0]), and the MAS returns to point 2.
   - 2.2 If one of the neighboring agents is present in the MAS, the new agent inquires about its position. Based on the knowledge of the absolute direction to this agent, the new agent calculates its position, and the MAS returns to point 2.

2.3 If two neighboring agents are present in the MAS, the agent determines their positions, takes the position of the first agent, calculates its own position, and determines the current absolute direction of its connection with the second agent. If it corresponds to the known absolute direction, the MAS returns to point 2.

2.4 If the determined absolute direction does not correspond to the known absolute direction, the agent determines its position by deriving it from the second agent. The agent then prompts the first agent to adjust its position to preserve the known absolute direction between them.

2.5 The first agent adjusts its position.

2.6 The first agent verifies the absolute directions to its potential neighbors, and if they all match the known absolute directions to them, the MAS returns to point 2.

2.7 Otherwise, it prompts the first of the conflicting neighboring agents to adjust its position so that the absolute direction towards it aligns with the known absolute direction and proceeds to point 2.6.

Definition 1 (*bounding circle BC*)**.** Let G be a planar graph and C be a graph circle. If each edge of the circle C has the maximal number Cn calculated by the equation:

$$Cn = ((RDN^{out} - RDN^{in} + 8) \mod 8) \tag{1}$$

Then C is called the ***bounding circle***.[3]

Absolut directions are fundamental elements in map sketching. We encode the directions from the input data using numbers.[4] The mapping of relative directions into natural numbers is shown in Table 1, which shows the basic eight directions that we obtain, i.e., (F – forward, FL - forward-left, L – left, BL – backward-left, B – backward, BR - backward-right, R – right, FR – forward-right).

RD	Encoding	RD	Encoding
F	0	B	4
FL	1	BR	5
L	2	R	6
BL	3	FR	7

Table 1. Directions and their numerical representation.

[3]RDN^{out} is Relative Direction Number (RDN) of outgoing incident edge and RDN^{in} represents RDN of the edge entering into the node.

[4]See more in [1].

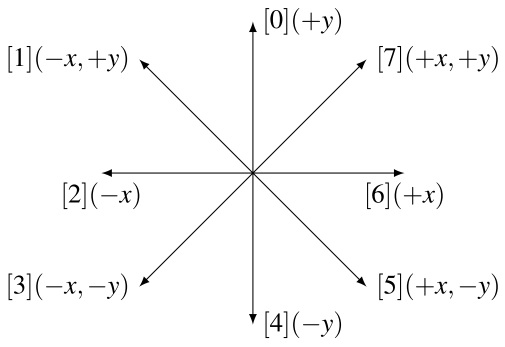


Figure 1. Absolute directions.

The *absolute direction* is related to a specific graph and is not aligned with the *cardinal directions* (North-South, East-West). In the middle of the compass rose, you can imagine an agent approaching from direction *4*, then its direction *F* is labeled with the number *0*. Because each agent can arrive at a given location from a different direction, a unification of directions through rotation occurs before graph rendering, ensuring a consistent absolute direction for all agents.[5]

3. Description of our Multiagent System.

To implement the proposed strategy, we chose a multi-agent system containing three types of agents: Gate Keeper, Broker, and Node agent.

The *Gate Keeper* is the unique entity within this system. It is the controlling element for the entry of *Node* agents into the system. Its task is to coordinate the addition of Node agents based on predefined *bounding circles* contained in its list. Additionally, the Gate Keeper keeps list of agents that have already entered the system to prevent the re-entry of the same agent. In this way, the Gate Keeper ensures that each Node agent is added to the system exactly once.

Therefore, the Gate Keeper has a pool of all agents, each with an already assigned unique identifier, such as the node number it represents. The Gate Keeper selects individual agents from this pool for entry into the system based on the order of the inserted bounding circle..

The *Broker* is another agent in the system, and like the Gate Keeper, there is only one Broker. It stores the complete topology of bounding circles of nodes in a labeled planar graph, where the labeling represents the direction between individual points. The Broker conveys this information to the newly arrived Node agent and waits until all Node agents arrange their positions to preserve the desired directions between them, stabilizing the system. The Broker then prompts the Gate Keeper to admit the next agent.

This system's last type of agent is the *Node* agent. This type of agent represents points on the bounding circle, and its task is to establish connections with neighboring Node

[5]The description of how directions are unified can be found in [1].

agents to create a predefined topology while preserving the directions between them. After connecting, the Node agent sets, or adjusts, its coordinates to represent the correct point in the defined topology. Depending on the solution's state regarding positions, this agent can take one of four states: WAITING, MOVING, PLACED, UNPLACED. For better orientation and understanding, agents will be further described by symbolic names (e.g., AgentA, AgentB, etc.).

Creating the topological graph begins with dividing Node agents into circles, which we then try to place gradually. The first Node agent added is placed at the origin (0,0) and is in the PLACED state. The next agent added to the system (being in the MOVING state) and adjacent to the first one in a given direction calculates its position relative to the first Node agent and places itself. The calculation is done through communication with the placed agent, inquiring about its coordinates. This is possible because the Node agent attempting to place itself always receives information from the Broker, which has an overall overview of the situation, about neighboring agents. With this information, the Node agent can communicate with them, calculate its position based on the acquired data, and place itself (changing its state to the PLACED state). If a Node agent does not have any neighboring agent in the system from which to calculate its position, it will wait in the system, meaning it will not be placed (remaining in the UNPLACED state).

Whenever a new Node agent attempts to place itself at the calculated position, it must check whether the path between it and its neighbor intersects with another edge connecting already placed agents.

When the incoming Node agent has two neighboring agents between which it must be placed, it must first calculate its position as an intersection based on the positions of the neighboring nodes and the directions to them. The calculated position is then submitted to the Broker to verify any path collisions. The Node agent places itself at the calculated position if there is no collision. If a collision occurs during the addition of a new Node, a request is sent to the already placed neighbor to adjust its position, and the Node agent tries to place itself again.

This is how Node agents communicate and adjust their positions to avoid path collisions. In cases where a neighbor, or both neighbors, cannot move for some reason, the new Node remains unplaced and waits (is in state UNPLACED). As mentioned, the Broker is responsible for checking intersections. It knows all nodes in the system, their positions, and directions, allowing it to determine whether the paths between nodes intersect or not.

In this way, distances and positions between individual Node agents are adjusted, resulting in a topological graph that corresponds to the topology of the input planar graph at the end.

3.1. Rules of behavior for agents in the system.

In this chapter, we summarize the behavior of individual agents. We do not provide detailed mathematical background here to avoid overwhelming the reader with technical details. When it comes to computing the movement of agents on a 2D plane, basic mathematical formulas and directions are utilized to determine the direction in which the agent should move.

3.1.1. GateKeeper

The Gate Keeper deploys Node-type agents into the system and checks whether a given Node is already present in the system. It has stored bounding circles to which individual Node agents are added.

- The first Node enters the system as stationary.
- If a Node wants to enter the system, it must be there exactly once.
- If a message about a stabilized system comes from the Broker, another Node can be added.

3.1.2. Broker

The Broker has stored bounding circles, their topology, and directions between nodes. It monitors the active neighbors of new Node agents on the specified circle. It validates edges between Node agents to prevent intersections.

- If the first agent wants to position itself, the Broker assign to it the position (0,0).
- A message with the name and address of the new Node is received. The Broker stores it and sends the Node a list of active neighbors on the circle.
- A message is received from the Node with its state (PLACED/UNPLACED). The Broker saves the state and waits until all Node agents are in this state. Then, it sends a message about the stabilized system to GateKeeper.
- If the Broker receives two positions from any Node agent, it checks all paths between active agents. It determines whether the path between these two positions intersects with another path.

3.1.3. Node

The Node agent enters the system only with its name and address. It obtains additional information from the Broker or its neighbors.

- When entering the system, the Node contacts the Broker with its address and name.
- If the Broker sends a list of neighbors, the Node saves them and contacts each.
- If the response from the neighbors includes positions, the Node calculates its position and sends it to the Broker for validation.
- If the Broker responds with VALID, the Node places itself at the calculated position.
- If the Broker responds with INVALID, the Node sends a message to one of its neighbors to adjust its position.
- If a message is received from this neighbor stating that the adjustment was not possible, the Node sends a request to another neighbor to adjust its position.
- If messages from all neighbors indicate that the adjustment was impossible, the Node sets its state to UNPLACED and sends it to the Broker.

- If it receives a message from its neighbor to adjust its position, the Node moves its position in the specified direction from its second neighbor.

Each agent utilizes a set of rules to choose actions and communicate with other agents. Thanks to this division, adding an agent to the system is straightforward. The only limitation arises from the need to verify the entire circle that contains the respective agent whether it is not necessary to adjust already placed nodes.

Communication

Communication is crucial in a Multi-Agent System (MAS) because without it, interactions among individual agents would vanish, and the system would degrade to purely reactive agents. The Agent Communication Language (ACL) is standardized to transmit data and physical messages. It is a structure composed of various components that specify the syntax and meaning of the message content, message parameters such as sender and receiver, and the type of communicative intention.[6]

Sample message between agents when sending positions to each other:

```
{
  "type": "position information",
  "sender": "AgentA",
  "receiver": "AgentB",
  "content": {
    "x": 0,
    "y": 2
  }
}
```

4. Case Study

In the following example, the creation of a map is demonstrated, consisting of three circles that are successively added, thereby forming a topological graph. The process of creating the topological graph is illustrated in the images below.

The input for this example is a list of circles. The notation of the circle list is in a format that always includes the number of the circle; below is a list of neighbors in that circle and a number C expressing the absolute direction from Node A to Node B. The triplet of data is written in the form ([A, B], C). An overview of absolute directions is shown in Figure 1.

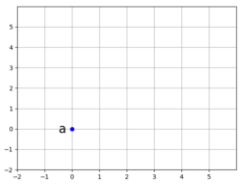
Bounding circles:

```
C0:             ([b,m],4)       ([l,k],1)       ([b,c],3)       ([k,g],2)
([a,p],7)       ([m,i],2)       ([k,g],2)       ([j,d],0)       ([l,p],3)
([p,l],7)       ([i,h],2)       ([g,e],0)       C2:             ([l,k],1)
([o,l],3)       ([h,a],0)       ([e,j],0)       ([p,a],3)
([c,o],3)       C1:             ([o,l],3)       ([a,f],0)
([b,c],3)       ([d,b],6)       ([c,o],3)       ([f,g],0)
```

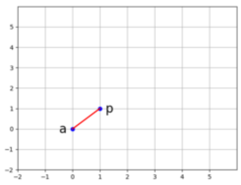
[6]More information about ACL can be found in [9],[10], [11].

According to the input of circles, GateKeeper will create a list of agent names, which will be sent to the system to be placed. In this example, the resulting list is as follows: [*a*,*p*,*l*,*o*,*c*,*b*,*m*,*i*,*h*,*d*,*k*,*g*,*e*,*j*,*f*].

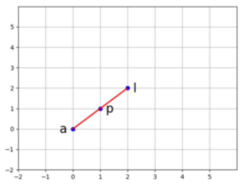
In the system, the first agent, denoted as *a*, is added, and it is stationary with the position (0,0) (see Fig. 2a). The system stabilizes, and GateKeeper allows the entry of another agent, *p*. It sends its address and name to the Broker. The Broker responds with a list of active neighbors, in this case, the address of Node *a*, its name, and the direction to it. Node *p* then contacts *a* and calculates its coordinates based on *a*'s coordinates and the direction, resulting in (1,1) (see Fig. 2b). Node *p* sends these coordinates to the Broker for validation. The Broker finds no collision, so Node *p* can be placed at the calculated position. Another node agent, *l*, is added in the same way as *p* (see Fig. 2c).



(a)



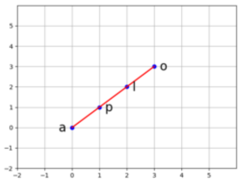
(b)



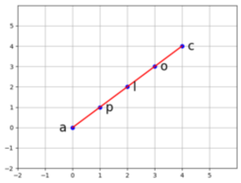
(c)

Figure 2. Adding the first nodes.

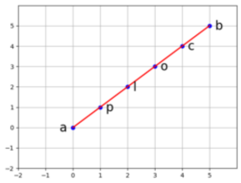
GateKeeper step by step introduces additional agents into the system. These agents communicate with their neighbors and settle into calculated valid positions, as depicted in Figure 3.



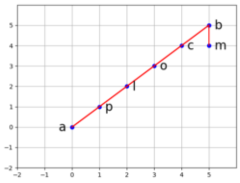
(a)



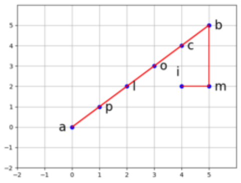
(b)



(c)



(d)



(e)

Figure 3. Adding additional nodes.

The description continues until the agent *h* (see Fig. 4a) joins the system. After connecting, it realizes its neighbors are Nodes *i, a*. It attempts to calculate the intersection but finds none. Therefore, *h* sends a message to Node *i* to move since Node *a* is stationary. Node *i* moves away from Node *m* but encounters a collision with Nodes *l, o*. Node *i* then sends a message to *m* to move as well. This process repeats until a valid intersection exists for Node *h* at coordinates (0, -1) (see Fig. 4c). The first circle is drawn, and the process continues for additional circles. The visualization of Node movements is depicted in Fig. 4.

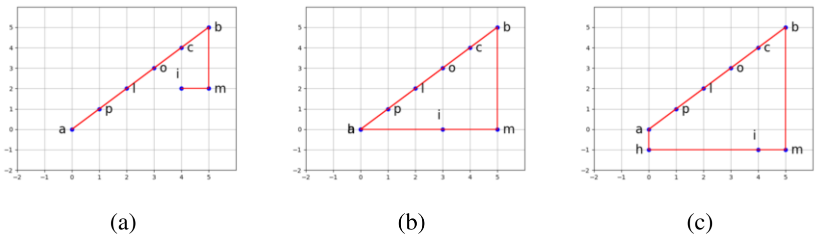


Figure 4. Node adjustments.

The system continues, and GateKeeper introduces additional nodes in the same manner. In the case of Node *j*, Node *d* must move to create an intersection between *d* and *e* for the placement of Node *j* (see Fig. 5d). After the movement (see Fig. 5e), the final node agent *f* calculates its intersection and settles onto it (see Fig. 5f). At this point, GateKeeper has an empty list of agents, and there are no more Nodes to let them for placement. The system is now complete.

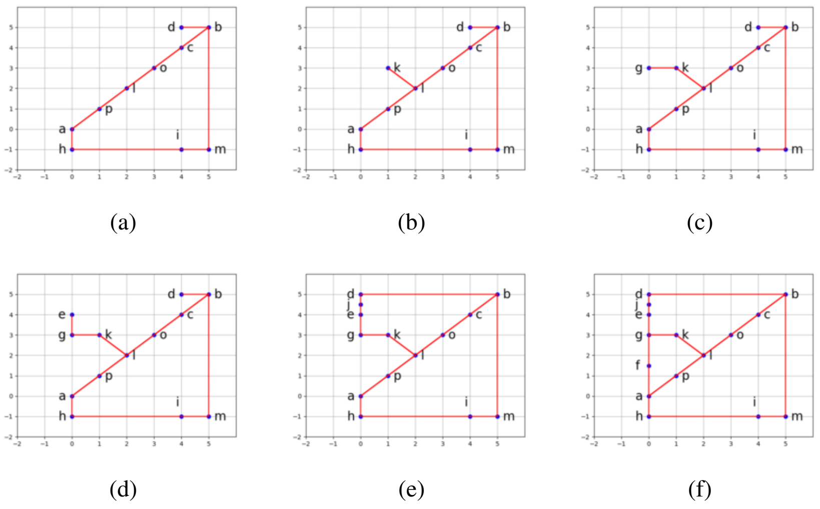


Figure 5. Adding final nodes.

The functionality of the proposed system was tested on a local machine, where each agent shares the same IP address but differs only in the port number. As a result, communication was not affected by physical network elements. Due to this setup, communication speed was in the range of milliseconds, and there were no losses of individual packets. However, the system is configured so that each agent waits for a response or acknowledgment of message delivery. Therefore, each agent waits for *50* milliseconds, and if there is no confirmation of receipt or response, the agent repeats the message transmission.

When running a multi-agent system on multiple machines, communication may take longer. In such cases, agents wait for message confirmation in the order of seconds. The agent resends the message if there is no response within that time frame. This adjustment accounts for the potential latency introduced by communication across multiple machines.

5. Conclusion

This paper outlines an algorithm that transforms a planar graph representing connections between places into a resulting topological graph. From the input data, we identify bounding circles, which are progressively added to the draft and subsequently merged into the overall topological graph. The entire process uses a multi-agent system, where individual agents represent the graph's nodes (places mentioned in narratives). Thanks to communication among agents, the process is distributed, allowing for easy monitoring of the construction of the entire sketch map. The system was implemented in the Python programming language.

Acknowledgements

This research has been supported by a Grant from SGS No. SP2024/108, VŠB - Technical University of Ostrava, Czech Republic, "Application of Formal Methods in Knowledge Modelling and Software Engineering VII".

References

[1] Marek Menšík, Petr Rapant, and Adam Albert. Algorithm for generating sketch maps from spatial information extracted from natural language descriptions. In *Frontiers in Artificial Intelligence and Applications*, volume 380, pages 239–252, Amsterdam, 2024. IOS Press.

[2] Martina Číhalová and Marek Menšík. Rules for converting natural language text with motion verbs into til-script. In *Frontiers in Artificial Intelligence and Applications*, volume 364, page 159 – 168, Amsterdam, 2023. IOS Press.

[3] Marek Menšík, Adam Albert, Petr Rapant, and Tomáš Michalovský. Heuristics for spatial data descriptions in a multi-agent system. *Frontiers in Artificial Intelligence and Applications*, 364:68–80, 2023.

[4] Michael J. Wooldridge. *Reasoning about Rational Agents*. Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge, 2000.

[5] Boda Ning, Qing-Long Han, Zongyu Zuo, Lei Ding, Q. Lu, and Xiaohua Ge. Multiagent systems in mobile robots and smart grids. *IEEE Transactions on Industrial Informatics*, 2023.

[6] Chuang Wang, Zidong Wang, Qing-Long Han, Fei Han, and Hongli Dong. Leader-follower-based particle swarm optimization algorithm for oilfield pipeline network. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

[7] F. Derakhshan and Shamim Yousefi. Applications of multiagent systems in healthcare and medical systems. *International Journal of Distributed Sensor Networks*, 2019.

[8] Hongtian Chen, Oguzhan Dogru, S. K. Varanasi, Xunyuan Yin, and Biao Huang. Process monitoring in wastewater treatment systems with multiagent systems. *IEEE Transactions on Cybernetics*, 2024.

[9] Fipa communicative act library specification. `http://www.fipa.org/specs/fipa00037/SC00037J.html`, 2002. Accessed: 2024-02-01.

[10] Ernesto German and Leonid Sheremetov. Specifying interaction space components in a fipa-acl interaction framework. In Mehdi Dastani, Amal El Fallah Seghrouchni, João Leite, and Paolo Torroni, editors, *Languages, Methodologies and Development Tools for Multi-Agent Systems*, pages 191–208, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] Marie Duží, Martina Číhalová, and Marek Menšík. Communication in a multi-agent system based on transparent intensional logic. In *Mendel 2011 : 17th International Conference on Soft Computing*, pages 477–485, Brno, 2011. Vysoké učení technické v Brně.