Spectral Clustering in Rule-Based Algorithms for Multi-Agent Path Finding

Irene Saccani[1], Kristýna Janovská[2] and Pavel Surynek[2]
[1]*University of Parma, Prama, Italy*
[2]*Czech Technical University in Prague, Prague, Czechia*

Keywords: Multi-Agent Path Finding, Rule-Based Algorithms, Spectral Clustering, Biconnected Components.

Abstract: We focus on rule-based algorithms for multi-agent path finding (MAPF) in this paper. MAPF is a task of finding non-conflicting paths connecting agents' specified initial and goal positions in a shared environment specified via an undirected graph. Rule-based algorithms use a fixed set of predefined primitives to move agents to their goal positions in a complete manner. We propose to apply spectral clustering on the underlying graph to decompose the graph into highly connected component and move agents to their goal cluster first before the rule-based algorithm is applied. The benefit of this approach is twofold: (1) the algorithms are often more efficient on highly connected clusters and (2) we can potentially run the algorithms in parallel on individual clusters.

1 INTRODUCTION

Multi-agent path finding (MAPF) is a task of navigating multiple agents $\{a_1, a_2, ..., a_k\}$ from their initial positions to given goal positions so that agents do not conflict with each other. The task often uses a graph theoretical abstraction where the environment is modeled as an undirected graph $G = (V, E)$ with at most one agent per vertex. Agents in this graph theoretical abstraction move across edges from their initial vertices specified via $s_0 : A \rightarrow V$ to their goal vertices specified via $g : A \rightarrow V$.

MAPF represents a major abstraction for motion planning of multiple robots. The applications of MAPF include warehouse logistics, coordination of multiple UAVs, traffic optimization, or navigation of multiple characters in computer games (Ma and Koenig, 2017).

We focus on the so-called rule-based algorithms for multi-agent path finding (Luna and Bekris, 2011; Surynek, 2009) where agents are moved in the graph via a-priori defined set of movement rules. These rules often regard the current configuration of agents in vertices of G as a permutation in which the rule makes local transformation. The advantage of these algorithms is their speed and scalability for large numbers of agents when compared to search-based techniques. On the other hand rule based algorithms do not generate optimal solutions with respect to commonly used objectives.

We propose two novel modifications that built on top of the two existing rule-based MAPF algorithms. We introduce a hierarchy in which we first decompose the underlying graph into clusters of high connectivity on which the rule-based algorithms are known to work well. Then the agents are moved intro their goal clusters and finally the specific rule-based algorithm finishes the final configuration of agents in the within the goal cluster without interacting with other clusters.

2 BACKGROUND

In this section we summarize the major existing rule-based algorithms for MAPF: Push-and-Swap (Luna and Bekris, 2011) and BiBOX (Surynek, 2009; Surynek, 2014).

2.1 Push-and-Swap

The Push-and-Swap algorithm is a method of solving MAPF in sub-optimal, yet efficient and complete manner. The algorithm consists of two basic primitives - one of them being Push and the other one being Swap. In Push, an agent moves along its shortest

Saccani, I., Janovská, K. and Surynek, P.
Spectral Clustering in Rule-Based Algorithms for Multi-Agent Path Finding.
DOI: 10.5220/0012206800003543
In *Proceedings of the 20th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2023) - Volume 1*, pages 258-265
ISBN: 978-989-758-670-5; ISSN: 2184-2809

path to its goal location and while doing so, pushes agents blocking its path out of their way, forcing them to clear a vertex.

In some cases though, a simple push may not be possible and thus the second primitive Swap must be employed. In the Swap primitive, two neighbouring agents swap their positions. This requires the agents to move to a part of graph where the swap is possible. A suitable part of a graph is a vertex with the degree of at least 3 and its neighbour.

This vertex has to have at least two free neighbouring vertices. If this is not fulfilled, the algorithm may try to free the neighbours of agents occupying them. After the swap takes place, all agents are back to their original positions by reversing the sequence of moves before the swap has been made. If the swap is not possible to perform, the MAPF instance is deemed unsolvable for Push-and-Swap [1].

2.2 BiBOX

The BiBOX algorithm assumes input MAPF where the underlying graph $G = (V,E)$ is bi-connected and there are exactly $|V| - 2$ agents (for fewer agents either fake agents can be used to fill up the graph or the algorithm can be slightly modified to deal with more vacant vertices). The rules of the BiBOX algorithms via which it moves agents into their goal positions significantly depends on the properties of bi-connected graphs.

A graph $G = (V,E)$ is bi-connected if for any two distinct vertices $u, v \in V$ there exist two disjoint paths that connect u and v (alternatively we can say that u, v belongs to a cycle in G). For graphs fulfilling this definition it holds that a so called *ear decomposition* can be applied to them. That is, each bi-connected graph $G = (V,E)$ can be constructed starting with an initial cycle by adding so called *ears* to the currently constructed graph. Assume to have $G_i = (V_i,E_i)$ at hand, a graph at the i-th step of the ear decomposition, adding an ear is represented by introducing a path H_i consisting of fresh internal vertices say $u, u_1, u_2, \ldots, u_{n_i}, v$ where $n_i \in \mathbb{N}_0$ and $u, v \in V_i$, that is we have $G_{i+1} = (V_{i+1}, E_{i+1})$, where $V_{i+1} = V_i \cup \{u_1, u_2, \ldots, u_{n_i}\}$ and $E_{i+1} = E_i \cup \{\{u, u_1\}; \{u_1, u_2\}, \ldots \{u_{n_i}, v\}\}$.

The BiBOX algorithm first determines an ear decomposition of the input graph G. Let us denote this ear decomposition $C, H_1, H_2, \ldots, H_m$, where C is the initial cycle and H_i are the individual ears. The time

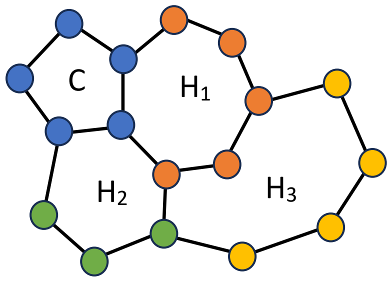

Figure 1: An illustration of the ear decomposition with the initial cycle C and three ears H_1, H_2, and H_3.

complexity of determining the ear decomposition is $O(n^2)$ where $n = |V|$ ().

Then the algorithm proceeds according to the ear decomposition starting with the last ear H_m into which agents are moved to their goal positions. Once all agents reach their goals in H_m they will no longer move and the last ear H_m can simply be ignored in the following stages of the BiBOX algorithm.

Moving agents into their goal positions within the ear H_m is done in a stack like manner. Assume that $H_m = \{u, u_1, u_2, \ldots, u_{n_m}, v\}$. We start with agent whose goal is in u_{n_m} and move it u to u_1 and then rotate the ear once so that the agent appears in u_1. Similarly we continue with an agent whose goal is in u_{n_m-1}. Again move the agent to u and rotate the ear once. We continue in the same way with the remaining agents whose goals in in H_m. If it happens that the agent we need to move into u already resides in H_m we need to rotate it out of the ear and rotate the ear back to restore the situation. Eventually we have all agents in their right positions in the ear.

Then the algorithm proceeds in the same way with the ears H_{m-1}, H_{m-2}, ... until the initial cycle C remains. For all the above process only one empty vertex in necessary. The second empty vertex will be used later.

At this stage all agents with their goals in C can be regarded as a permutation that may differ from the goal permutation represented by agents' goal positions in C. BiBOX now utilizes two empty vertices to swap pairs of agents in C using a vertex from H_1 as a parking place. Being able to swap the agents the goal permutation can be reached.

3 RELATED WORK

It is important to note that the motivation for the development of sub-optimal algorithms for MAPF is that the optimal variant with respect to any common objective, such as the number of moves, the

[1]There is a modification of the Push-and-Swap algorithm called Push-and-Rotate (de Wilde et al., 2014) that eliminates some issues of Push-and-Swap. For our purposes however, the original Push-and-Swap is applicable.

makespan, or the sum-of-costs is NP-hard (Ratner and Warmuth, 1990).

There exist multiple alternatives to Push-and-Swap and BiBOX. These alternatives include other sub-optimal rule-based algorithms that use different movement primitives (Wang and Botea, 2011; Khorshid et al., 2011; Krontiris et al., 2013). All these algorithms in some sense originate from the works on *graph pebbling* (Wilson, 1974; Kornhauser et al., 1984). Some of these algorithms use decomposition of the input graph similarly as we do such as (Ryan, 2008) where sub-graphs of various types use special movement primitives suitable for a given sub-graph type.

Significant progress has been made in optimal solving of MAPF, where the most commonly adopted objective is the sum-of-cost, the sum of unit costs of all actions of agents including the wait action. There exist algorithms derived from the standard A* with various improvements to tackle multiple paths (Silver, 2005; Standley, 2010). More modern approaches to MAPF rely on variants of the *conflict-based search* algorithm (Sharon et al., 2012) from which sub-optimal variants were derived (Barer et al., 2014; Li et al., 2021).

Sub-optimal algorithms can be derived also from compilation-based techniques for MAPF that translate the question of existence of a solution to a different formalism such as Boolean satisfiability (SAT) (Surynek et al., 2017).

It is important to note, that search-based algorithms are often incomplete as well as their sub-optimal variants.

4 CONTRIBUTION

The original studies where the BiBOX algorithms and the Push-and-Swap algorithm are introduced, argue that the algorithms are suitable for graphs densely populated by agents. On the other hand the algorithms use little of the structure of the underlying graph G. In this work we are trying to utilize the structure of the graph more while keeping the advantages of rule-based algorithms for dense cases.

We propose hierarchical variants of BiBOX and Push-and-Swap that first decompose the input graph into highly connected components and then run the specific rule-based algorithm on the components. This has twofold effect: first, the high connectivity of the component enables higher efficiency of the rule-based MAPF algorithm, and second, the algorithm can be run in parallel on individual components.

Specifically we use **spectral clustering** (Luo et al., 2003) to find suitable components of the graph. This is a numeric method based on calculation of eigenvalues from the adjacency matrix of G. The advantage of numeric methods for graph clustering is that we can easily fine tune the output clustering via the change of numeric parameters of the clustering algorithm to fit our needs. These parameters can be, for example, the number of clusters or the parameters used in the chosen clustering algorithm.

The first new algorithm is our hierarchical version of BiBOX:

1. clusters of the input graph are determined via spectral clustering
2. each cluster is decomposed into bi-connected components (the spectral clustering should be tuned so that this step is rather small or does almost nothing, i.e. clusters should be near bi-connected)
3. move agents via modified Push-and-Swap into their goal bi-connected component
4. reach the final configuration of agents in each bi-connected component by BiBOX

We will call this algorithm **Hierarchical BiBOX** or **ChiBOX** in short.

The second algorithm is a simple modification of the previous one. We alternatively run Push-and-Swap to reach the final configuration in each cluster (not bi-connected component, hence the construction of bi-conncted components is skipped). We will call this algorithm **Hierarchical Push-and-Swap** or **Chi-Push-and-Swap** in short.

4.1 Spectral Clustering and Bi-Connected Components

We want to decompose the graph in several sub-graphs. A simple way to achieve that is by using **Spectral Clustering** (Luo et al., 2003; Von Luxburg, 2007). The Spectral Clustering is based on numeric methods hence can be easily parameterized to obtain various clusterings according to our needs.

Firstly, we compute the optimal number of clusters. This can be done in a number of ways based on the structure of the graph. For example, if the graph has a regular structure, with nodes that have a nearly equal degree, the number of clusters can be decided based on the number of nodes, as the connectivity and the number of nodes of each cluster will be more or less the same. For random biconnected graph an approach that takes into account the different degrees of the nodes and finds a balance between the dimension

Algorithm 1: Spectral Clustering.

```
  Input   : $A$ = adjacency matrix
  Output: clusters, c
1 $L \leftarrow NormalizedLaplacian(A)$;
2 $\lambda \leftarrow eigenvalues(L)$;
3 $v \leftarrow eigenvectors(L)$;
4 $v, \lambda \leftarrow sort(v, \lambda)$;
5 $k \leftarrow maxGap(\lambda)$;
6 U ← matrix where the columns are the first $c$
    eigenvectors;
7 X ← normalized rows of U;
8 clusters ← K-means(X, c);
9 return clusters, $c$
```

of each cluster and the connectivity inside the clusters and between the clusters is needed. In this paper the number of clusters is computed using the *Eigengap Heuristic*. To do so, we find the normalized Laplacian L of the adjacency matrix of the graph. Then we computed the eigenvalues and the eigenvectors of L and sorted them based on the eigenvalues. The ideal number of clusters is given by the index of the maximal difference between one eigenvalue and the next. Having found the optimal number of clusters c, we create the matrix U where the columns are the first c eigenvectors and normalize the rows of U. Finally, we use the rows to compute the clusters using a clustering algorithm such as K-means (Ng et al., 2001).

In figure 2 a grid graph and a random bi-connected graph are colored based on the cluster of each node obtained from the spectral decomposition.

Proposition 1. *The total time complexity of spectral clustering is* $O(n^3) + O(cnT)$.

Proof. The time complexity of finding the eigenvalues of the Laplacian is $O(n^3)$ where $n = |V|$. The time complexity of K-means is given by $O(cnT)$ where n is the number of samples (again $n = |V|$ in our case), T is the number of iterations and c is the number of clusters. ■

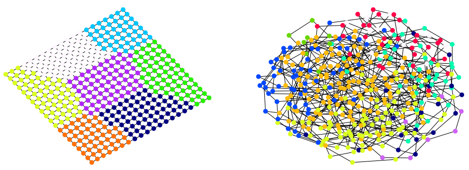
Figure 2: A 20x20 grid graph and a random bi-connected graph are colored based on spectral clustering.

Then, we create a new graph where every node represents a cluster and two clusters are linked if two of their nodes are neighbours, this graph is called a *cluster graph*. In figure 4, the cluster graphs of the grid graph and the random bi-connected graph can be seen.

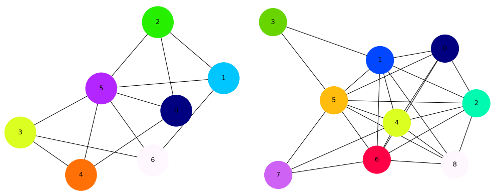

Figure 3: Cluster graphs of graph at figure 2.

Each cluster is then analysed and divided into its bi-connected components so that we are can apply the BiBOX algorithm later. A new graph is then built, in which each node represent a different bi-connected component and an edge is added between two bi-connected components if there exists, in the original graph, an edge linking nodes of those components.

The search of bi-connected components on a cluster i is achieved using the Depth First Search algorithm that has the complexity of $O(n_i)$ where n_i is the number of nodes of the cluster, so the search of bi-connected components in all the clusters has the total complexity of $O(n)$, where $n = |V|$ the number of nodes in the original graph G. Searching the edges of the new graph has a complexity that is $O(n^2)$. Therefore, this part of the algorithm has a complexity that is $O(n) + O(n^2)$.

In figure 4 we can see the bi-connected component graphs. The color of each node shows the cluster of each bi-connected component.

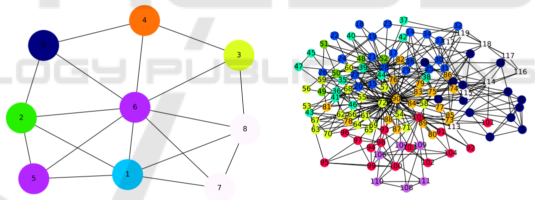
Figure 4: Bi-connected component graphs of graphs in figure 2.

We can see that the grid graph, will have clusters with a more regular structure and that have a smaller number of bi-connected components, whereas a random graph will have clusters with a higher number of bi-connected components.

4.2 Combining Spectral Decomposition, Push-and-Swap and BiBOX

The output of Spectral Clustering consists of two parts. The first is a graph where each node represents one cluster - one bi-connected graph. This graph serves as input for the Push-and-Swap algorithm (Luna and Bekris, 2011) which acts as device to move agents between clusters. The second part of the output is a list of all decomposed clusters. These sub-graphs are bi-connected graphs connected to at least one other bi-connected sub-graph.

In Push-and-Swap, multiple agents can share a start vertex and a goal vertex. This is because all vertices represent sub-graphs between which agents move. A goal of an agent in Push-and-Swap is a particular vertex representing a sub-graph, but without a concrete position in that sub-graph. The purpose of this algorithm is to get all agents into their goal sub-graphs.

This setting then serves as an input for the BiBOX algorithm (Surynek, 2009). Multiple instances of BiBOX have to be run for each of n vertices resulting from Spectral Clustering. By decomposing the original graph into disjoint sub-graphs, it is possible to work on a reduced part of the problem at a time and it is not necessary to consider all of the agents in the original graph.

By dividing a graph into several smaller disjoint instances, we reduce not only the graph itself, but mainly the number of agents traversing this graph.

In a densely occupied graph, agents may traverse longer paths in order to avoid collisions with obstacles presented by other agents. Dividing that graph not only reduces staging necessary for planning paths for many agents at once, but also reduces operations agents perform to get in their target loop as this has already partially been performed by Push-and-Swap.

Disjoint BiBOX instances also offer the possibility of parallelization of this part of the algorithm, as no traversions of agents between these instances is further necessary.

One of rules of BiBOX is that at least two vertices of its input graph have to be unoccupied (Surynek, 2009). For n decomposed bi-connected sub-graphs this means that in total there have to be $2n$ free vertices in the original graph. This algorithm is therefore usable for $|V| - 2n$ agents, where $|V|$ is the number of vertices in the original graph and n is the number of decomposed components.

The advantage of spectral clustering is that is has various numeric parameters through which fine tune the output clustering for specific MAPF algorithm. Such fine tuning is difficult with the standard graph search algorithms.

4.3 Push-and-Swap Modification

To serve this setting more effectively, Push-and-Swap was modified in following ways.

Firstly, individual nodes in the cluster graph represent bi-connected components - sub-graphs of the original graph G, not single nodes. As agents reach their goal node, they can be ignored as they move further into their respective sub-graph (cluster). This makes the resolve operation, in which agents, that have previously reached their goal and were moved due to pushing and swapping another agents had to be placed back at their goal positions unnecessary. As we can now ignore agents at their goal position during the whole course of the algorithm, agents at their goal will never be moved to a different sub-graph (cluster).

Different sub-graphs are connected by one node of each respective sub-graph. Let's call this node the entrance node. Entrance nodes have to be freed from agents at their goal sub-graphs in order for agents to pass between sub-graphs without a collision. Each sub-graph is a bi-connected graph and as per BiBOX's precondition has to contain at least two free vertices (Surynek, 2009). Therefore maintaining the entrance node free from completed agents should be possible at all times if there is less than two agents present who do not have this component set as their goal component.

To achieve this, we propose a new function Component-Push. In this function, all the agents in a respective sub-graph (cluster or bi-connected) are pushed further into that sub-graph, so that an agent occupying the entrance node will free it while not moving away from its goal sub-graph. As this operation does not aim to move agents to their individual goal positions inside their sub-graph, a push operation is enough to clear the entrance node. All of the agents consider their neighbouring node in opposite direction than the entrance node as their goal. Therefore in this operation, all of the agents present in the sub-graph move by one node. This way they prepare their initial positions for BiBOX algorithm, which commences after all of the agents have reached their goal sub-graphs.

Algorithm 2: Operation Component-Push.

```
Input  : a = agent number,
         C = component,
         p = agent position,
         u = used vertices
Output: True if operation was successful
1 $neighbour \leftarrow$ get-neighbour$(C, p, u)$;
2 $u \leftarrow u \cup neighbour$;
3 if $occupying\text{-}agent(neighbour)$ != $\emptyset$ then
4     if $\neg$component-push
      (occupying-agent(neighbour), C,
      neighbour, u) then
5         return False
6 assignment$[a] \leftarrow neighbour$;
7 path $\leftarrow$ path $\cup$ assignment;
8 return True
```

Algorithm 2 describes the recursive operation Component-Push. The input of this function consists of an agent number belonging to agent to be pushed, the component - sub-graph where this is taking place, position of an agent that is to be pushed and a list of vertices, that have already been used to push.

Firstly, all neighbours of a's vertex that fulfill requirements (they have not yet been used to push) are found. The current position of a is then added to u. Then the push itself is performed. If the neighbouring vertex is not free, it is to be freed with another Component-Push. If this sequence of actions is not possible to perform, *False* is returned. After freeing *neighbour*, position of a is changed to it. List *assignment* stores information about the latest positions of all agents. Finally, the new assignment is added to *path*, which represents all paths of all agents from the beginning of the algorithm to the current step.

Agents in their starting arrangement are located in a component, that may differ from their goal component. To move to a different component, they can too perform operation Component-Push, only they now have their goal position set as their entrance node. As if they were to push themselves further into the component, they push other agents only around that respective component, never between components.

This version can also possibly take advantage of these individual components when a graph without at least one vertex of degree of at least 3 is presented. It can possibly perform the swap by temporarily using a vertex of a component with a free vertex.

Proposition 2. *The time complexity of the ChiBOX algorithm and the Chi-Push-and-Swap algorithm is* $O(n^3)$.

Proof. The time complexity of building clusters in the spectral clustering is dominated by the time complexity the underlying rule-based MAPF algorithms, BiBOX or Push-and-Swap, that have the time complexity of $O(n^3)$ for $n = |V|$. ■

5 PRELIMINARY EXPERIMENTS

In this section we comment on preliminary experiments that we obtained with our implementation of ChiBOX and Chi-Push-and-Swap in Python.

5.1 Spectral Clustering and Bi-Connected Components

The decomposition into bi-connected components has been tested on *NetworkX* standard graphs such as the grid graph and the complete graph, and on bi-connected random graphs. To assure that even highly connected graphs had enough clusters for the decomposition to be used for the subsequent use on the algorithm, a minimum value for the number of clusters has to be imposed. In figure 5 we can see how different minimum values for the number of clusters affect the spectral decomposition. In this case a random bi-connected graph with 200 nodes is created in which the degree of every node can vary between 2 and 10. Performing the *Eigengap Heuristic* on all the eigenvalues (first image) says that the ideal number of clusters would be 1. This is not useful for our algorithm, so we can decide to search the maximal difference between eigenvalues with index $i \geq min$, forcing the number of clusters to be higher than the minimum value chosen (min). In the second image of figure 5 $min = 3$, so the algorithm finds the maximal difference between eigenvalues for $i \geq 3$ that is $i = 5$. The other images represents clusters with $min = 7$ ($i = 7$) and $min = 9$ ($i = 59$). We can notice that the last case, in which the number of clusters is 59 is also not useful for our algorithm. In this case a balance has to be found between the minimum and the maximum value of the number of clusters.

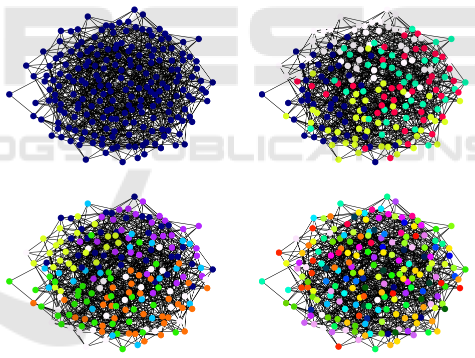
Figure 5: Random bi-connected graphs with nodes colored based on spectral clustering. The minimum value for the number of clusters is 1, 3, 7, 9.

5.2 Comparing BIBOX with Chi-Push-and-Swap and ChiBOX

Preliminary experiments have shown that the main advantage of our approach lies in safe parallelization of BiBOX on individual sub-graphs. As these bi-connected sub-graphs are disjoint, agents in different sub-graphs can move in parallel and not wait after an agent in a different sub-graph has taken a step (although this is not possible in Chi-Push-and-Swap, where agents move sequentially). On a simple 12 vertex bi-connected graph decomposed into 4 clusters (4 bi-connected components), each having 3 vertices, 4 agents found paths using Chi-Push-and-Swap and then parallel ChiBOX in 13 steps, while in original BiBOX a sequential path was found with 15 steps. Therefore Push-and-Swap and parallel ChiBOX are able to find a shorter path compared to BiBOX.

As the same clusters may in reality be connected by more than one edge in different vertex, redundant edges are ignored in the input of Chi-Push-and-Swap. Preferable edges to preserve are those which connect higher-degree vertices in the original graph.

Another advantage of this approach is the way loops are created. In the original graph, 7 loops were created for BiBOX, including the original cycle. However due to the spectral decomposition, only 4 loops were necessary - one original cycle per biconnected component. Therefore this approach may aid in simplifying loop decompositions.

6 DISCUSSION AND CONCLUSION

We proposed a novel approach for the rule-based algorithms for multi-agent path finding. We first decompose the input graph into highly connected components via the spectral clustering method, a numeric method based on calculation of eigenvalues of the adjacency matrix of the input graph. Then agents are moved to their goal clusters and after this the specific rule-based algorithm is executed on individual clusters to move agents to their goal vertices within the cluster. We implemented this new method on top of the BiBOX and Push-and-Swap algorithms, we call the new variants ChiBOX and Chi-Push-and-Swap.

Our preliminary experiments indicate that the new methods are promising and can produce solutions that are better in the term of the number of moves than if the rule-based algorithm is applied directly on the unprocessed input graph.

For future work we plan to fine tune the spectral clustering method to produce clusters that are suitable for specific rule-based algorithms.

ACKNOWLEDGEMENTS

This research at the Czech Technical University in Prague has been supported by GAČR - the Czech Science Foundation, grant registration number 22-31346S.

REFERENCES

Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 961–962. IOS Press.

de Wilde, B., ter Mors, A., and Witteveen, C. (2014). Push and rotate: a complete multi-agent pathfinding algorithm. *J. Artif. Intell. Res.*, 51:443–492.

Khorshid, M. M., Holte, R. C., and Sturtevant, N. R. (2011). A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011*. AAAI Press.

Kornhauser, D., Miller, G. L., and Spirakis, P. G. (1984). Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 241–250. IEEE Computer Society.

Krontiris, A., Luna, R., and Bekris, K. E. (2013). From feasibility tests to path planners for multi-agent pathfinding. In Helmert, M. and Röger, G., editors, *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013*. AAAI Press.

Li, J., Ruml, W., and Koenig, S. (2021). EECBS: A bounded-suboptimal search for multi-agent path finding. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 12353–12362. AAAI Press.

Luna, R. and Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conference on Artificial Intelligence*.

Luo, B., Wilson, R. C., and Hancock, E. R. (2003). Spectral clustering of graphs. In *Graph Based Representations in Pattern Recognition, 4th IAPR International Workshop, GbRPR 2003, Proceedings*, volume 2726 of *Lecture Notes in Computer Science*, pages 190–201. Springer.

Ma, H. and Koenig, S. (2017). AI buzzwords explained: multi-agent path finding (MAPF). *AI Matters*, 3(3):15–19.

Ng, A., Jordan, M., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14.

Ratner, D. and Warmuth, M. K. (1990). Nxn puzzle and related relocation problem. *J. Symb. Comput.*, 10(2):111–138.

Ryan, M. R. K. (2008). Exploiting subgraph structure in multi-robot path planning. *J. Artif. Intell. Res.*, 31:497–542.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2012). Conflict-based search for optimal multi-agent path finding. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012*. AAAI Press.

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, 2005*, pages 117–122. AAAI Press.

Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press.

Surynek, P. (2009). A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation, ICRA 2009*, pages 3613–3619. IEEE.

Surynek, P. (2014). Solving abstract cooperative pathfinding in densely populated environments. *Comput. Intell.*, 30(2):402–450.

Surynek, P., Felner, A., Stern, R., and Boyarski, E. (2017). Modifying optimal sat-based approach to multi-agent path-finding problem to suboptimal variants. In *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017*, pages 169–170. AAAI Press.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17:395–416.

Wang, K. C. and Botea, A. (2011). MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *J. Artif. Intell. Res.*, 42:55–90.

Wilson, R. M. (1974). Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86–96.