In Proceedings of Robotics: Science and Systems (RSS), June 26-30, 2018. This is the manuscript of the authors.

GPU-Based Max Flow Maps in the Plane

Renato Farias Marcelo Kallmann
Computer Science and Engineering Department
University of California, Merced, CA, 95343
E-mail: {rfarias2,mkallmann}@ucmerced.edu

***Abstract*—One main challenge in multi-agent navigation is to generate trajectories minimizing bottlenecks in environments cluttered with obstacles. In this paper we approach this problem globally by taking into account the maximum flow capacity of a given polygonal environment.**

Given the difficulty in solving the continuous maximum flow of a planar environment, we introduce in this paper a GPU-based methodology which leads to a practical method for computing maximum flow maps in arbitrary two-dimensional polygonal domains. Once the flow is computed, we then propose a method to extract lane trajectories according to the size of the agents and to optimize the trajectories in length while keeping constant the maximum flow achieved by the system of trajectories.

As a result we are able to generate trajectories of maximum flow from source to sink edges across a generic set of polygonal obstacles, enabling the deployment of large numbers of agents optimally with respect to the maximum flow capacity of the environment. Our approach eliminates bottlenecks by producing trajectories which are globally-optimal with respect to the flow capacity and locally-optimal with respect to the total length of the system of trajectories.

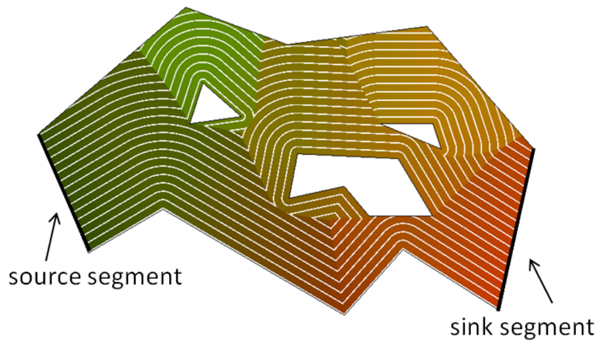


Fig. 1: This scenario illustrates a max flow map computed from a source edge to a sink edge. While this flow map has optimal flow capacity, path lanes are subsequently optimized in length according to the characteristics of the agents navigating the flow.

I. Introduction

The problem of optimally deploying multiple agents traversing a polygonal environment has important applications in many areas, for example, to control multiple robots in warehouses, to coordinate autonomous cars across narrow streets and to evaluate evacuation scenarios. While optimality can be defined by taking into account different variables such as energy, time, or distance travelled, in all cases the problem is difficult to be solved in a planar domain and is usually addressed in a discrete representation of the environment.

In this paper we introduce the approach of relying on the continuous maximum flow of a given environment and we present a new GPU-based method that allows us to compute the maximum flow and to represent it in a *max flow map*.

By computing the max flow map of the environment, bottleneck-free lanes can be determined leading to a system of trajectories able to deploy agents to safely reach their destinations. If a large quantity of agents is deployed, the system of trajectories will optimally guide all agents to reach the destination region. Here optimality is related to the maximum number of agents that can be deployed across the environment from a source polygonal entrance to a sink polygonal exit without creating bottlenecks.

We consider the input polygonal domain to be delimited by entrance and exit polygonal edges which function as the source and sink of the obtained max flow map. See Fig. 1. Once the flow map is computed, lane trajectories are extracted according to the size of the agents and optimized in length while keeping constant the maximum flow achieved by the system of trajectories.

As a result our method is able to generate trajectories of maximum flow from source to destination edges among a generic set of polygonal obstacles, guaranteeing that no bottlenecks are formed. Given the obtained flow optimality of the trajectories, when the system of trajectories is fully occupied by agents moving along them, no more agents can fit the environment without eventually creating bottlenecks. Agents can safely follow our computed trajectories without having to employ any complex local behavior strategies in order to reach the destination region.

Our trajectories are globally-optimal with respect to the flow capacity and locally-optimal with respect to the total length of the generated system of trajectories. Several simulations are presented demonstrating the superior performance of our method in deploying large quantities of agents across environments with obstacles.

II. Related Work

Multi-agent path planning is important for a variety of applications and the problem has been extensively studied in different contexts.

The vast majority of approaches developed to date are based on grid representations. In this case, the problem consists of finding trajectories for agents from given initial cells to given target cells in the environment grid, while avoiding cells marked as obstacles [7]. A popular approach to this problem is to plan paths individually for each agent and subsequently solve all conflicts. Different strategies for solving conflicts exist. For example, one approach is to recursively solve conflicts between pairs of agents [10]. A number of variations and extensions exist, such as integration with roadmaps for scalability to higher dimensional problems including articulated structures [3]. While finding optimal solutions for several versions of the multi-agent path finding problem is known to be NP-hard [15, 13], unlabeled variations can be solved in polynomial time [17, 11].

Discrete flow algorithms have clear applications to multi-agent path planning and the problem of computing maximum flows in a capacitated network graph has been very important in combinatorial optimization. The problem is commonly studied in textbooks and many polynomial-time algorithms exist. The connection between network flows and path planning has started to be investigated in a number of works; however, to date all previous works have been limited to investigations performed in discrete versions of the problem.

A Conflict-Based Min-Cost-Flow algorithm has been proposed to address the combined target assignment and path finding problem, where a min-cost max-flow algorithm on a time-expanded network is used to assign all agents in a single team to targets [6]. Yu and Lavalle [16] study the problem of computing minimum last arrival time and minimum total distance solutions for multi-agent path planning on graphs. Their formulation relies on discrete multi-commodity flow algorithms which address the problem of flowing different types of commodities through a graph network.

Heuristics for search-based algorithms that systematically explore the state space have also been proposed based on commodity flows [14], and multi-agent path planning has been investigated with graph network flows, with respect to goal replacement for paths when the goals are permutation invariant [17]. In the area of multi-agent simulation, crowd-flow graphs have been developed to distribute agents in an environment according to capacity information extracted from a harmonic field computed in the environment [1].

While these works clearly show that solving flow problems represents a powerful approach to address multi-agent path planning, no previous work has explored the use of a continuous flow formulation in order to be able to directly address planar environments described by polygonal boundaries. Such an approach is important in order to reach optimality guarantees in the Euclidean sense, and furthermore, to take into account specific geometric constraints (such as agent size) without simplifications.

While the generalization of the maximum flow problem to a continuous domain is clearly interesting, its computation is not obvious. Strang [12] describes an extension of the max flow-min cut theorem to continuous flows, showing that the maximum flow from sources to sinks in a planar domain is determined by the minimal cut, just like the discrete version of the problem. This result opens a direction for computing max flows in continua.

Mitchell [8] addresses the problem of actually constructing the min cuts and max flows in a clever approach based on the computation of Shortest Path Maps (SPMs) [9]. Polynomial-time algorithms are given for varied max flow scenarios involving source edges and sink edges in simple polygons. Similar to the calculation of SPMs, a continuous Dijkstra paradigm forms the basis for their algorithms, but in a specific form which solves the so called $0/1/\infty$ weighted regions problem. In this paper we follow this approach in order to achieve our proposed flow maps.

While a traditional CPU implementation of SPMs based on the continuous Dijkstra paradigm is still difficult to accomplish, in this paper we approach the problem with a new GPU implementation which greatly simplifies addressing the problem. The approach is based on the concepts introduced by Camporesi and Kallmann [2]. In this paper we extend this GPU approach to take into account polygonal edges as sources and the required $0/1/\infty$ weighted region formulation. By applying our extended SPM methods we are able to obtain a diagram expressing the desired flow map for the entire input environment.

While it is possible to generate SPMs via CPU-based methods, our GPU implementation was developed in order to achieve a practical approach to the problem by making use of the built-in features of the OpenGL rendering pipeline. Additional information of our implementation is available [4] and also demonstrates performances superior to some other approaches.

Contributions This paper proposes two main contributions. First, while previous works have been limited to discrete flow definitions, we introduce a method to compute maximum flow maps in continuous domains, relying on the insight of applying GPU rasterization techniques previously used for computing shortest paths with SPMs. Second, we demonstrate a simple method to optimize the flow trajectories with respect to their total length, while keeping the maximum flow capacity optimal.

III. Method Overview

The goal of our overall method is to be able to optimally generate paths for multiple agents traversing a polygonal environment cluttered with obstacles.

The input polygonal environment is delimited by a polygon $\mathcal{P}$ containing all obstacles of interest in its interior. We assume that agents will enter $\mathcal{P}$ from given source edges P_{src} and will exit the environment by crossing sink edges P_{snk}. In our formulation P_{src} and P_{snk} are polygonal lines which are pieces of the domain boundary $\mathcal{P}$.

While it is possible to consider multiple disconnected polygonal lines as sources and sinks, in this work we limit ourselves to a single polygonal line for each. Assuming P_{src} is left of P_{snk}, as in the example of Fig. 1, there are two additional polygonal lines between P_{src} and P_{snk} which appear at the bottom and top of the domain. We call these additional polygonal lines as P_{bot} and P_{top}. In this case the concatenation of P_{src}, P_{bot}, P_{snk} and P_{top} completely covers the domain boundary in counter-clockwise order.

With source and sink edges defined it is then possible to compute a max flow in $\mathcal{P}$. A *flow* in $\mathcal{P}$ is as a divergence-free vector field defined for every valid point in the interior of $\mathcal{P}$. The vector field of the flow provides directions that can be followed by agents in $\mathcal{P}$. We informally define a maximum flow in $\mathcal{P}$ as a flow maximizing flow directions crossing P_{snk} outwards, and with directions along P_{src} entering $\mathcal{P}$. Given that the flow is divergence-free a max flow provides a way to route agents from P_{src} to P_{snk} while maximizing the flow of agents exiting $\mathcal{P}$. While capacity constraints could be defined in terms of maximum magnitudes of flow vectors, in our application we consider a constant magnitude everywhere. Formal definitions of the above concepts are provided by Mitchell [8].

Our approach to compute a max flow for $\mathcal{P}$ however does not take into account the size of agents or the length of the generated trajectories when agents follow the flow. We then present a second step to address these additional aspects.

Considering that an agent traversing the environment is approximated by a circle of radius r, we present a method to extract *lanes* from the max flow such that each lane connects entrance points on P_{src} to exit points on P_{snk}, and to optimize them in length. The produced lanes will be of maximum flow, will minimize total length, and will have clearance r; that is, for each point in a lane its minimum distance to the boundary $\mathcal{P}$, the obstacles inside $\mathcal{P}$, or to other lanes, will be at least r.

The next sections detail the steps discussed above.

IV. Shortest Path Map

We start by describing our approach to compute Shortest Path Maps (SPMs), which are necessary for computing the proposed max flow maps. For additional details an extended exposition of our SPM computation method described in this section is also available [4].

SPMs are structures constructed with respect to one or more "source points" or "source segments", and that partition the space into regions that share the same sequence of parent points along the shortest path to the closest source. This means that an SPM encodes shortest paths to the sources for *all* points in a particular planar environment, and is thus useful for global navigation. Fig. 2 shows an example SPM computed for one of our test environments.

Let n_s source points $\{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_{n_s}\}$ be defined in the plane, such that $\mathbf{s}_i \in \mathcal{P}$, $i \in \{1, 2, ..., n_s\}$, and where $\mathcal{P} \subset \mathbb{R}^2$ defines a polygonal domain containing all sources and obstacles. We can also have n_l line segment sources $\{\mathbf{l}_1, \mathbf{l}_2, ..., \mathbf{l}_{n_l}\}$, such that $\mathbf{l}_i$, $i \in \{1, 2, ..., n_l\}$, consisting of two endpoints $\in \mathcal{P}$. A set of polygonal obstacles $\mathcal{O}$, with a total of n vertices, is also defined in $\mathcal{P}$ such that shortest paths will not cross any obstacles in $\mathcal{O}$.

Given source points and segments the respective SPM will efficiently represent globally-shortest paths, which are optimal collision-free paths from any point $\mathbf{p} \in \mathcal{P} - \mathcal{O}$, to either: 1) its closest source point $\mathbf{s}_i$ or 2) the closest reachable point on its closest segment source $\mathbf{l}_i$ (as will be discussed in section IV-B).

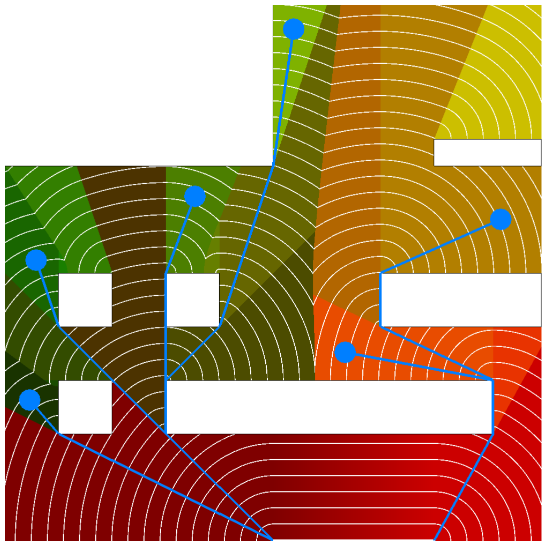


Fig. 2: Shortest Path Map of our Scenario 2 map. Contour lines represent points equidistant to the SPM's source segment, and the blue circles represent agents whose shortest paths are shown with blue lines.

We follow the approach of Camporesi et al. [2] and improve it in order to take into account polygonal edges as sources and to eliminate geometry approximation. Our solution is implemented using OpenGL Compute Shaders [4]. The approach is based on rasterizing "clipped cones" with apices placed at specific depths below source points and obstacle vertices, relative to the $z = 0$ plane, such that the final rendered result from an orthographic top-down view is the desired SPM. Each apex's z coordinate is equal to its distance to the closest source along the shortest path. When a cone is rasterized, the depth values of the affected pixels increase proportionally to their Euclidean distances to the apex, and thus the GPU's depth test will maintain only the closest ones to a source.

A. Algorithm

An array containing points with the xy coordinates of the n_s source points and n obstacle vertices is stored in the GPU. These points are referred to as "generator points", or simply "generators."

The GPU framebuffer stores the following information for the pixels: 1) the *red* and *green* channels store the xy coordinates of the pixel's current parent point in $\mathcal{P}$; 2) the *blue* channel stores the current known shortest path distance to the closest source; 3) the *alpha* channel is used as a flag to determine whether the pixel has yet to be rasterized by a generator. When the buffer is visualized, we zero out the *blue* channel so that we can visualize the xy coordinates of each region.

The SPM generation process consists of choosing a generator from the array and rasterizing its clipped cone $n_{total} = n_s + n$ times, such that each generator is processed once.

First, we determine which generator will be rasterized next by choosing the one with the smallest distance to source. This means that naturally the source points themselves will be the first ones to be picked, as their distance to source is 0. Points which have already been chosen before are not considered, and a non-source point cannot become a generator if it has not been reached by a previous cone because its correct distance to source is not yet known.

Second, we rasterize the clipped cone. This means that only the parts of the scene which have direct line-of-sight to the cone are rasterized, hence why it is "clipped." This visibility determination is part of the construction of the clipped cones, and is done before the actual rasterization. The pixels that are affected by the rasterization have direct line-of-sight to the generator point, so they calculate their Euclidean distance to it and add it to the generator's accumulated distance. If this sum is smaller than the current distance stored in the pixel (from the cone of a previous generator), then its $blue$ channel stores the new distance and its red and $green$ channels are updated with the generator's coordinates, making the generator the pixel's new parent point.

After all generators have been processed, the result in the framebuffer will be the desired SPM.

B. Line Segment Sources

To extend the SPM algorithm from source points to source segments, we must also process "elongated" cones that have apices which are segments rather than just points.

Let $\mathbf{l}_i$ be a source segment with n_{c_i} *critical points*, $n_{c_i} \geq 0$. These critical points come from the visibility set of segments of the critical graph, which will be explained in greater detail in Section V-A. Critical points are the points on the source segments onto which obstacle vertices can be projected (see Fig. 3).

Since critical points denote points on the segment where the visibility of the scene changes with respect to the segment, we must consider each of the sub-segments between the endpoints and the critical points independently. We consider that $\mathbf{l}_i$ is now made up of $n_{c_i} + 2$ points: its two endpoints plus all of the critical points along its length, if any.

Each pair of adjacent points makes up a sub-segment that is included in the SPM generation process as an elongated cone. In practice this means that the distance calculation of the pixels is slightly different when the generator is a segment, as it must determine whether it is closer to one of the endpoints or to a projected point somewhere inbetween. However, this is still just a point-to-segment distance calculation.

V. Max Flow

Given the polygonal domain $\mathcal{P}$, the obstacles in it, source edges P_{src} and sink edges P_{snk}, we can then compute a maximum flow from P_{src} to P_{snk}. Central to the approach taken in this paper is the intimate connection between the max-flow problem and its dual, the min-cut problem.

The min-cut of the environment represents a polygonal line cut, starting at P_{top} or P_{bot}, traversing the environment, and ending at the opposite boundary, P_{bot} or P_{top}, respectively. The total length of the sub-segments of the min-cut that are on the free space of the domain represents the main bottleneck of the environment, and defines the maximum possible flow from source to sink. Relevant to our work is the fact that the min-cut can be determined by accumulating distances from P_{bot} to P_{top} (or from P_{top} to P_{bot}) without considering the cost of traversing obstacles.

The needed distance accumulation is addressed by the $0/1/\infty$ weighted region problem, which is a limited case of the general Weighted Region Problem where only three weights exist: 0 (no cost), 1 (cost proportional to distance traveled), and ∞ (impassable region) [5]. With this formulation, when generating a SPM with source as P_{bot} or P_{top}, distances are accumulated from one boundary of the domain to the other without considering obstacles, which are seen as 0-cost regions rather than the ∞-cost regions they would be in a normal SPM.

The vector field defining our max flow will consist of the vectors orthogonal to the isolines of the generated SPM. As we will later see, the final flow lanes that will be used will be extracted directly from the generated flow and we will not need to explicitly extract the min-cut.

A. Critical Graph of the Domain

In order to compute the SPM of an environment with 0-cost regions, the *critical graph* [8] of the domain has to be computed first.

The critical graph of the domain contains essential information for the construction of the max flow map via the SPM algorithm, and is comprised of line segments between the boundary and the obstacles in the scene, or between the obstacles themselves. There are two distinct sets of line segments that make up the critical graph: a *visibility* set and a *flow propagation* set.

The visibility set of line segments serves to inform the SPM algorithm at which points along a polygonal line the visibility of the scene changes with respect to that polygonal line. It is derived from projecting all obstacle vertices onto the polygonal line of the domain boundary, either P_{bot} or P_{top}, that will be used to generate the max flow map. Assuming, for example, that we are using P_{bot}, the obstacle vertices with direct line-of-sight to P_{bot} project onto its segments as shown in Fig. 3. The construction of this part of the critical graph depends on the choice between P_{bot} and P_{top}, which themselves depend on P_{src} and P_{snk}.

The purpose of the flow propagation set of line segments is to inform us of the shortest segment that exists between the boundary and each obstacle, and between pairs of obstacles. This is crucial to correctly propagate flow distances throughout the scene, as will be explained in Section VI. The vertices from the entire boundary are projected onto all obstacle segments, and the vertices of all obstacles are projected onto the segments of all other obstacles. A line segment is added to the critical graph if this line segment is the shortest distance between them. An example is shown in Fig. 3. The construction of this part of the critical graph does not depend on the choice of source or sink, and therefore depends only on the configuration of the scene.

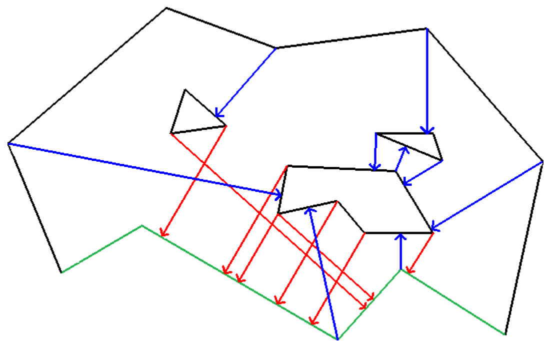


Fig. 3: The critical graph of an environment with three obstacles. In this example, P_{bot} (green) is the part of the boundary that will be used to generate the map, and therefore the vertices project onto it. The visibility set of segments is shown in red and the flow propagation set of segments is shown in blue. Arrows illustrate the direction of each vertex's projection.

VI. Max Flow Map

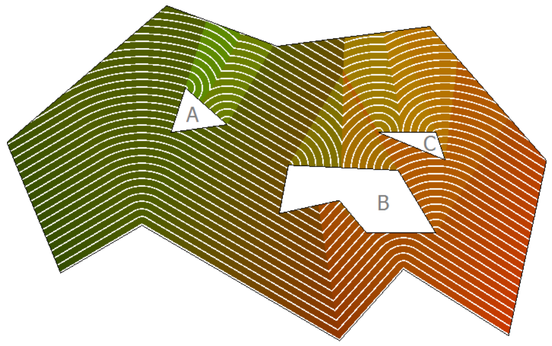
We compute our max flow map by using the SPM propagation approach as described in the previous section. The initial pre-processing step is to compute the critical graph of the scene, as explained in Section V-A.

In this section we choose P_{bot} to generate the max flow map. The generation process is illustrated in Fig. 4 and follows the three steps enumerated below:

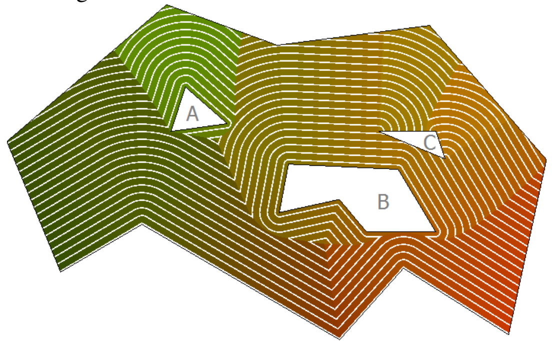
1) First, all of the segments of P_{bot} are set to be initial SPM segment sources and the SPM of the scene is computed.
2) Then, for each obstacle in the domain, the critical graph is used to find its smallest distance to the closest segment of P_{bot}. If an obstacle finds a new smallest distance, the SPM is updated, expanding all of the obstacle's segments again using this new distance. This step updates the SPM to take into account 0-cost transitions across the selected obstacle.
3) Finally, the previous step is repeated until none of the obstacles find new smallest distances. The resulting map is the max flow map.

By computing the max flow map and querying it, we are able to determine in constant time which direction each agent should move so that maximal flow is achieved. The max flow map is a vector field where each pixel now stores a direction rather than a single parent point. Each direction is set to be orthogonal to the SPM distance field, i.e., the max flow vector field will store vectors tangent to the white SPM isolines shown in the figures.

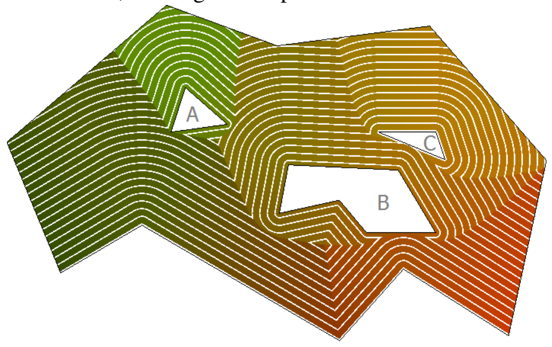
For practical use the obtained map can then be divided into "lanes", which are corridors that originate from P_{src} and follow the flow field until reaching P_{snk}. By following these lanes agents will move towards the sink in an orderly



(a) SPM generated from an initial scene configuration with three obstacles A, B, and C, and the corresponding critical graph as shown in Fig. 3.



(b) Obstacles A and B find new smallest distances, and expand again with them, altering the map.



(c) Obstacle C finds a new smallest distance through obstacle B and expands again, finalizing the max flow map.

Fig. 4: Example steps to generate a max flow map.

fashion that guarantees maximal flow. Agents of course have a certain size and lanes must have enough clearance to prevent collisions with obstacles and with other agents. A lane determination process is therefore necessary.

We assume that each agent can be represented by a circle of radius r. A candidate lane whose minimum distance to any obstacle segment and to $\mathcal{P}$ is less than r is considered to be invalid, otherwise it is valid.

Our lane determination procedure works as follows. We take points along P_{src} from an extreme endpoint towards the other extreme endpoint in order to generate candidate lanes. If a candidate lane is found to be invalid, we advance by a small increment Δ along P_{src} and try again. In our scenarios we have set Δ to be the height of a pixel. If a valid candidate lane is found, we advance by $2r$ because we know adjacent lanes must be spaced by at least $2r$. Because of this and the fact that lanes run parallel to each other, we do not need to check for collision with previously-accepted valid lanes. With this procedure we determine the maximum possible number of valid lanes from P_{src} to P_{snk}.

A possible optimization to the above procedure is to directly compute the total incremental space needed when an invalid lane is found, instead of performing increments of Δ size. This is possible to obtain from the largest obstacle penetration depth encountered in an invalid candidate lane. However the approach of testing by small increments is simpler and was effective in our scenarios.

The choice of P_{bot} or P_{top} as the origin of the SPM generates two different max flow maps which almost certainly have different configurations of lanes. However, because the max flow is determined by the min cut, and the min cut is dependent only on the scene, this choice has no effect on the capacity of the maximal flow.

VII. Length Optimization of Flow Lanes

Utilizing the max flow map described in the previous section and assigning agents to all the possible lanes guarantees a continuous max flow of agents through an environment. However, depending on the layout of the scene and the relative sizes of the source and sink, the generated lanes may be inefficient with respect to the path they take through the scene, taking extremely long detours when shorter, more direct paths are available. We present here a post-processing optimization process to reduce this inefficiency.

For each lane, we randomly choose a pair of points p_1 and p_2 along its path, and check to see if the segment $\overline{p_1p_2}$ is a valid "shortcut." This is only true if $\overline{p_1p_2}$ does not intersect with any other occupied lane or obstacle segment and keeps its minimum distance to all obstacles and $\mathcal{P}$ as at least r, and to all other lanes as at least $2r$.

This optimization can be applied individually to any lane, in any order, and repeated any number of times. In practice, we start the process with the last assigned lane and work our way backwards to the first. This is because if P_{src}'s length is less than the min cut of the environment, the lanes will not be able to use all available space up to the side opposite of the one that generated the max flow map, and therefore the last lane tends to have the most open space to work with. As shortcuts are accepted and lanes are shortened, they also free up new space for subsequent lanes.

The optimization does not change the original lane assignment, it only shortens the lanes instead of searching new ways through the environment, so this optimization does not guarantee a globally-optimal configuration of lanes length-wise nor does it alter the maximal flow. However, by iterating enough times, it converges to a locally-optimal solution. The effect of this process on the lanes of our test environments is illustrated in Figure 5.

VIII. Results and Discussion

In order to illustrate the benefits of using our max flow trajectories we have produced several multi-agent simulations in different environments.

In each simulation we define P_{src} at the top of the domain boundary and P_{snk} at the bottom, then proceed to construct three types of navigation environments:

1) The first type is based on the SPM computed with P_{snk} as source, such that agents will follow their shortest paths to P_{snk}. We call this SPM as SPM$_{snk}$. Paths are the shortest possible but several bottlenecks occur which are handled with simple collision avoidance between the agents. This SPM$_{snk}$ case is not to be confused with the SPM pass used to construct the max flow map (next type).

2) The second type uses our max flow map of the environment, which is computed with the $0/1/\infty$-SPM formulation starting from P_{bot}. The flow map provides directions to agents placed anywhere in the covered regions of the environment. Lanes respecting the size of the agents are retrieved from P_{src} to P_{snk} and used to guide the agents. The lanes are optimal with respect to the flow capacity but their lengths can be further optimized.

3) In the third type the lanes obtained from the flow map are optimized leading to a system of trajectories with minimized total length while still achieving the max flow of the environment.

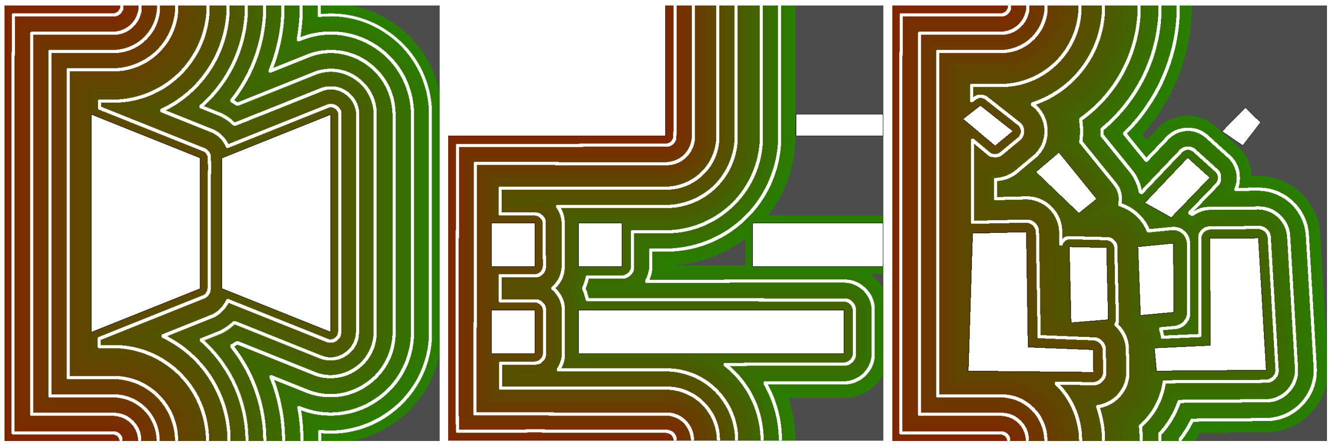
In each environment type we repeatedly spawn agents at the beginning of each lane whenever there is space for them, as the agents use either the SPM$_{snk}$, the max flow map, or the max flow map with optimized lanes to navigate towards the sink. Whenever an agent reaches the sink, it is removed from the environment. Fig. 6 shows a snapshot of the simulation running on scenario 3.

The simulations ran for 60 seconds, measuring the minimum, maximum, and average lengths of the paths computed and the number of agents that were able to reach the sink during that time, as can be seen in Table I. The SPM$_{snk}$ consistently computed the shortest paths in every environment, which is to be expected since it gives the globally shortest path for each point. However, fewer agents were able to reach the sink during the simulation. When too many agents try to follow their shortest paths to the sink, bottlenecks emerge that slow down the majority of their progress.

The environment types relying on the max flow, as expected, despite having longer overall lane lengths, were better for coordinating the movement of agents throughout the environment. No bottlenecks were created, and so the max flow map was able to make 2 or sometimes close to 3 times as many agents reach their destination. Also, agents using the max flow map do not require collision avoidance behavior.

The accompanying video to this paper[1] demonstrates the path optimization procedure and full simulations running in three different environments for each type of navigation strategy.

[1]available at http://graphics.ucmerced.edu/publications.html


(a) original lanes

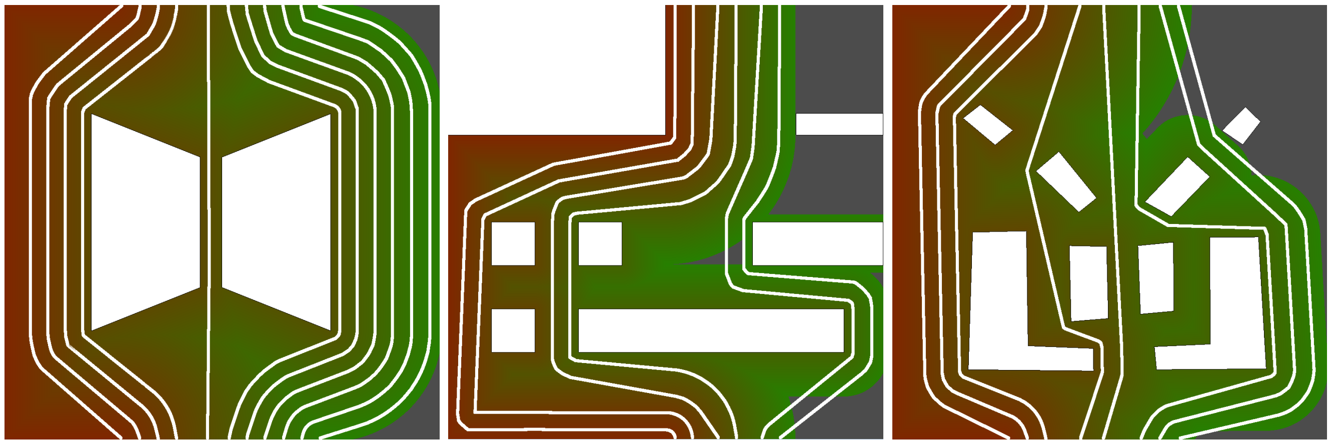

(b) optimized lanes

Fig. 5: Scenarios 1 (left), 2 (middle), and 3 (right).

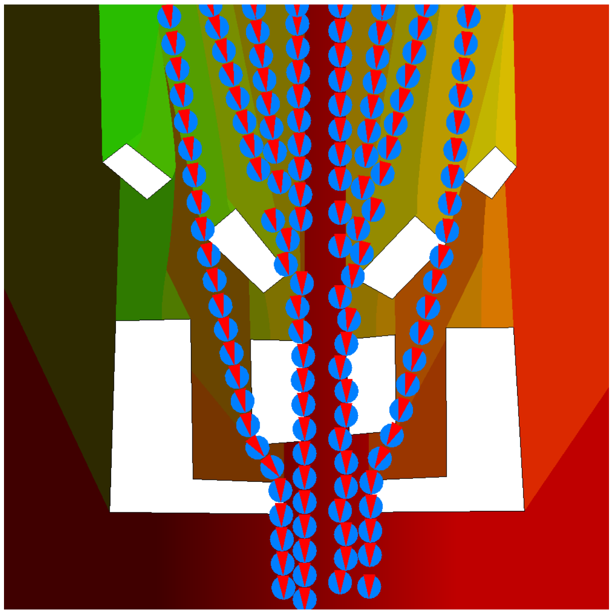
	Scenario 1				**Scenario 2**				**Scenario 3**			
Method	**min**	**max**	**avg**	**n**	**min**	**max**	**avg**	**n**	**min**	**max**	**avg**	**n**
SPM$_{snk}$	1.97	2.02	1.99	453	2.24	2.28	2.26	214	1.97	2.03	2.00	441
Max Flow	2.57	3.92	3.06	1053	3.30	4.39	3.89	553	2.74	4.31	3.20	766
Max Flow (optimized)	1.97	2.52	2.39	1165	2.65	3.61	3.09	611	1.98	2.89	2.40	843

TABLE I: The results of simulations where agents continuously spawn at the beginning of their assigned lanes whenever there is enough space for them, and then navigate towards the sink, for a duration of 60 seconds. Columns *min*, *max*, and *avg* refer to the minimum, maximum, and average path/lane lengths computed for the scene, respectively, and *n* is the total number of agents that were able to reach the sink in the allotted time. The three scenarios are illustrated in Fig. 5.

Our results clearly show the benefits of computing optimal flow trajectories for deploying large numbers of agents across generic polygonal domains. Because the computed flow is optimal, no better solution can be found in terms of numbers of agents per second reaching the exit sink polygonal line. Our optimization of trajectories also ensures that each agent will follow a short path to the exit; however, our current method does not guarantee a globally-optimal solution in terms of overall path length of all trajectories, because we do not evaluate all combinatorial possibilities of assigning trajectories, or portions of trajectories, to different corridors. Such an extension is left for future work. An additional promising direction for future work is taking into account disconnected source and sink polygonal lines.

IX. Conclusion

We have introduced in this paper a new method to compute max flow maps capturing the maximum flow capacity of given generic polygonal domains. The proposed method is able to determine bottleneck-free lanes that lead to a system of trajectories optimally guiding a large number of agents to reach a destination exit of the environment. The presented simulations demonstrate that our approach can dramatically increase the number of agents that successfully navigate towards the goal


(a) SPM$_{snk}$

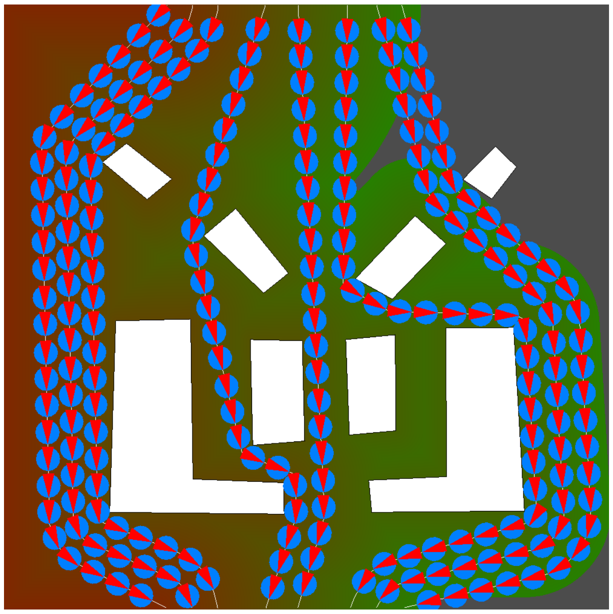

(b) Max flow with optimized lanes

Fig. 6: Snapshots of simulations on scenario 3. Despite both having the same amount of space and 8 lanes to start with, the SPM$_{snk}$ only permits 4 agents to reach the exit at a time, while the max flow map permits all 8 to do so.

exit of the environment in a given time frame.

The proposed approach introduces a methodology for taking into account continuous flows in polygonal domains, opening new research avenues in flow-based multi-agent path planning.

Acknowledgments

This research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-17-1-0463. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. The authors also thank Prof. Joseph S. B. Mitchell for several discussions on the topic of this paper.

References

[1] Adam Barnett, Hubert P. H. Shum, and Taku Komura. Coordinated crowd simulation with topological scene analysis. *Computer Graphics Forum*, 35(6):120–132, September 2016.

[2] Carlo Camporesi and Marcelo Kallmann. Computing shortest path maps with GPU shaders. In *Proceedings of the Seventh International Conference on Motion in Games*, MIG '14, pages 97–102, New York, NY, USA, 2014. ACM.

[3] Andrew Dobson, Kiril Solovey, Rahul Shome, Dan Halperin, and Kostas E. Bekris. Scalable asymptotically-optimal multi-robot motion planning. In *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS), Los Angeles, CA, USA, December 4-5, 2017*, pages 120–127, 2017.

[4] Renato Farias and Marcelo Kallmann. Improved shortest path maps with GPU shaders. E-print arXiv:1805.08500 [cs.GR], May 2018. URL https://arxiv.org/abs/1805.08500.

[5] L. Gewali, A. Meng, J. S. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, pages 266–278, New York, NY, USA, 1988. ACM.

[6] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. In *Proceedings of the 2016 International Conference on Autonomous Agents &#38; Multiagent Systems*, AAMAS '16, pages 1144–1152, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems.

[7] Hang Ma and Sven Koenig. AI buzzwords explained: multi-agent path finding (MAPF). *AI Matters*, 3(3):15–19, 2017.

[8] Joseph S. B. Mitchell. On maximum flows in polyhedral domains. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, pages 341–351, New York, NY, USA, 1988. ACM.

[9] Joseph S. B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3(1):83–105, 1991.

[10] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219(C):40–66, February 2015. ISSN 0004-3702.

[11] Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion planning for unlabeled discs with optimality guarantees. In *Robotics: Science and Systems*, 2015.

[12] Gilbert Strang. Maximal flow through a domain. *Mathematical Programming*, 26(2):123–143, Jun 1983.

[13] Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In *AAAI*, 2010.

[14] Jiri Svancara and Pavel Surynek. New flow-based heuristic for search algorithms solving multi-agent path finding. In *ICAART (2)*, pages 451–458. SciTePress, 2017.

[15] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, pages 1443–1449. AAAI Press, 2013.

[16] Jingjin Yu and Steven M. LaValle. Planning optimal paths for multiple robots on graphs. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3612–3617, 2013.

[17] Jingjin Yu and Steven M. LaValle. Multi-agent path planning and network flow. In *Algorithmic Foundations of Robotics X*, pages 157–173, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.