Noname manuscript No.
Autonomous Robots 2020, to appear. This is the manuscript of the authors.

Planar Max Flow Maps and Determination of Lanes with Clearance

Renato Farias and Marcelo Kallmann
Computer Science and Engineering Dept.
University of California, Merced, CA
{rfarias2,mkallmann}@ucmerced.edu

Received: date / Accepted: date

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 present in this paper a GPU-based methodology which leads to practical methods for computing maximum flow maps in arbitrary two-dimensional polygonal domains. Once a flow map representation is obtained, lanes can be extracted and optimized in length while keeping constant the flow capacity achieved by the system of trajectories. This work extends our previous work on max flow maps by presenting a clearance-based flow generation method which takes into account the size of the agents at the flow generation phase. In this way we ensure that the maximum possible number of lanes with the needed clearance is always obtained, a property that was found to not be always obtained with our previous method.

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 utilizing the maximum flow capacity of a continuous description of the environment and eliminating bottlenecks.

1 Introduction

The problem of optimally deploying multiple agents traversing a polygonal environment has important applications in many areas, as 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 parameters such as energy, time,

Address(es) of author(s) should be given or distance traveled, 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 present a method for agent deployment among obstacles based on computing the continuous maximum flow of a 2D environment. In this case we address computing solutions based on disjoint lanes which are optimal with respect to the maximum flow of agents traversing the environment. Our overall approach is based on GPU rasterization techniques which allow us to compute maximum flows from polygonal representations and to represent them in a *max flow map* discretized in a frame buffer in the GPU.

Our overall approach was recently introduced [5] and in this extended description of our methods we include an alternative type of max flow map, a clearance-based max flow map, which generates max flows specifically for achieving the maximum possible number of lanes with a given clearance. We present examples showing that this property is not always observed with our original, clearance-insensitive, max flow map method. The new method achieves this property by incorporating during the flow generation a technique suggested for generating *integer* max flows [12]. We therefore describe in this paper two types of max flow maps: one insensitive to the clearance needed by agents, and another specific for a given clearance. We also include definitions for the types of flows that are represented in each method.

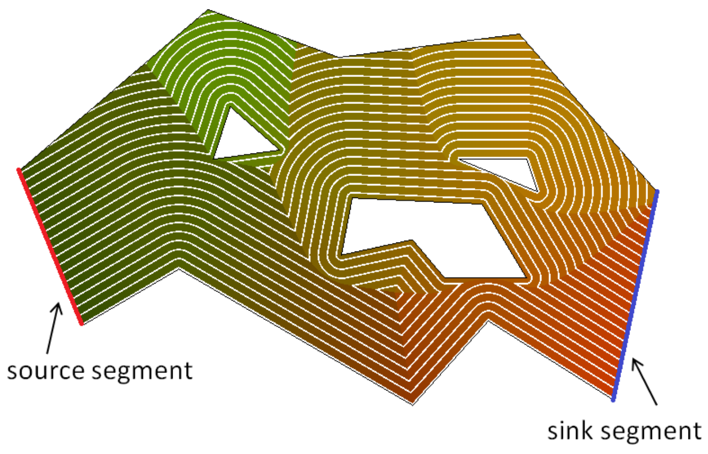


Fig. 1: Example max flow map from a source edge to a sink edge. While this flow map has optimal flow capacity, path lanes are subsequently extracted taking into account the required agent clearance and then optimized in length.

Our methods produce bottleneck-free lanes which can be used to safely deploy and guide agents across a cluttered environment. If a large quantity of agents is deployed, the system of lanes 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. See Figure 1 for an example. Once a 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 methods are able to generate lanes of maximum flow from source to destination edges among a generic set of polygonal obstacles. When the system of lanes is fully occupied by agents, no more agents can fit the environment without eventually creating bottlenecks. Agents can safely follow our computed lanes without having to employ any complex local behavior strategies in order to reach the destination region. In terms of length, our lanes are locally-optimal with respect to the total length of all lanes.

Simulations are also presented demonstrating the superior performance of our method in deploying large quantities of agents across environments with obstacles, when compared to having agents following their shortest paths to the destination.

2 Related Work

Our work develops a new approach to address multi-agent navigation in cluttered environments which is based on finding the maximum number of valid disjoint lanes that can be routed from an initial polygonal source segment to a polygonal target segment.

The addressed problem has some resemblance to the well-known k-disjoint shortest paths problem in graphs [2], which seeks to find the k pairwise disjoint shortest paths connecting given initial graph nodes to corresponding target graph nodes. Our formulation however is quite different in which it addresses a continuous polygonal environment, with polygonal edges considered as entrances and exits, and addressing required clearance constraints. Another difference is that there is no labelled target per agent, and instead agents can be routed to any point in the exit segment. Our proposed approach solves this problem by computing the continuous max flow of the input environment and then extracting paths from it. While our solution is of max flow, the total path length is optimized only locally.

The literature review below makes an overview of the more generic Multi-Agent Path Planning (MAPP) problem, analyzes previous uses of max flow algorithms in MAPP problems, and finally reviews the continuous max flow problem which is the approach taken in this work.

2.1 Multi-Agent Path Planning

The Multi-Agent Path Planning (MAPP) problem is related to planning paths for agents from their initial positions to target positions. It is an important problem for a variety of applications and the problem has been extensively studied in different contexts.

One important characteristic of methods to solve the MAPP problem is that, in order to find a solution when one exists, time has to be discretized and the employed search procedure has to take into account the time component. Our approach however focuses on finding disjoint paths such that searching in the time component is not needed. While our approach can be seen to be less generic in the sense that it does not solve agent coordination to reach specific goal points, the proposed method addresses the spatial reasoning problem of maximizing the flow of agents across an environment, and is better suited for optimizing areas with high traffic of agents among obstacles. Once paths (or lanes) of maximum flow are computed, an arbitrary number of agents can be deployed by simply following the lanes in order to optimally traverse the environment.

The vast majority of MAPP 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 [10]. 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 [14]. Another is to have agents to follow paths which may have conflicts, and to let each agent address the conflicts reactivelly in response to the local environment and nearby agents [19]. A number of variations and extensions exist, such as integration with roadmaps for scalability to higher dimensional problems including articulated structures [4]. While finding optimal solutions for several versions of the multi-agent path finding problem is known to be NP-hard [22, 17], unlabeled variations can be solved in polynomial time [20, 15].

2.2 Use of Flow Algorithms in Multi-Agent Path Planning

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 an important problem 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 graph is used to assign all agents in a single team to targets [9]. Yu and Lavalle [21] 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 [18, 8], and multi-agent path planning for goals that are permutation invariant has been addressed with graph network flows [20].

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, previous work has used discrete max flow algorithms applied to a time-expanded representation. No previous work in multi-agent navigation has explored the use of a continuous flow formulation in order to directly generate lanes and at the same time 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.

2.3 Continuous Max Flows

While the generalization of the maximum flow problem to a continuous domain is clearly interesting, its computation is not obvious. Strang [16] 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 [11] 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) [13]. While no implementations are presented, 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 the 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 it is not straightforward to generate max flows and SPMs via CPU-based methods, our GPU-based techniques represent a practical approach to achieve implementations solving these problems. Our underlying GPU-based approach takes advantage of the built-in rasterization features of the OpenGL rendering pipeline in order to propagate costs during the construction of our maps. While in this paper we focus on the generation of max flow maps, we also describe SPMs and we include in the Appendix a self-contained description of our SPM construction method. A complete description with additional details, extensions and benchmarks against other approaches for building SPMs is available in our previous work [6]. We are not aware of any other implementations to compute maximum flows.

2.4 Contributions

This paper proposes two main contributions. First, we introduce methods to compute maximum flow maps for polygonal domains relying on the insight of applying GPU rasterization techniques previously used for computing shortest path maps. We then address the new problem of extracting paths with clearance from max flows, presenting a specific flow construction method that takes into account clearance, and addressing methods for lane extraction and total length minimization.

3 Definitions and Overview

Let the input polygonal environment be delimited by a polygon $\mathcal{P}$ containing all obstacles of interest in its interior. The set of polygonal obstacles is denoted by $\mathcal{O}$. We are considering the situation where agents will enter $\mathcal{P}$ from given source edges P_{src} and exit the environment by crossing sink edges P_{snk}, while not colliding with any obstacles in $\mathcal{O}$. In our formulation P_{src} and P_{snk} are polygonal lines which are pieces of the boundary of the domain $\mathcal{P}$. We also consider that P_{src} and P_{snk} are connected polygonal lines.

Assuming P_{src} is left of P_{snk}, as in the example of Figure 1, there are two additional polygonal boundaries 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 possible to define the max flow problem in $\mathcal{P}$. A few variations on the definitions have been presented in the literature. Mitchell [11] defines the max flow problem as computing a vector field $\sigma : \mathcal{P} \rightarrow \mathbb{R}^2$ that maximizes $v = \int_{P_{snk}} \sigma \cdot \mathbf{n}\, ds$, subject to: div $\sigma = 0$ and $|\sigma| \leq c$ in $\mathcal{P}$.

In the above definition, vector $\mathbf{n}$ is the outward unit vector normal to P_{snk}, v is the value of the flow σ, and c is a capacity constraint function that can be defined to limit the magnitude of the vector field. Flow conservation comes from the divergence-free constraint, which also implies that flow-in equals flow-out. A variation of this definition includes the additional constraint $\sigma \cdot \mathbf{n} = 0$ on the boundary of obstacles [12]. There might be different maximum flows for one given environment.

For our navigation applications we have found that having directions at the sink edges orthogonal to the sink edges is not necessary for maximizing the number of outgoing agents that can exit the environment. It is enough that directions are outgoing, i.e., that $\sigma \cdot \mathbf{n} > 0$. Considering a more generic setting, we would also only be interested in maximizing the flow with respect to outgoing directions at the sink edges which are reachable, by following the flow, from a point in a source edge. These adaptations are addressed by function f in our proposed Definition 1. This definition also specifies that the magnitude of the vector field is always 1 when it is defined, or 0 in regions where the flow is not useful.

Definition 1 (Max Flow.) *A max flow in $\mathcal{P}$ is a vector field $\sigma : \mathcal{P} \to \mathbb{R}^2$, that maximizes $\int_{P_{snk}} f(\mathbf{p})\, ds$, subject to:* div $\sigma = 0$ *in $\mathcal{P}$ and $|\sigma| \in \{0, 1\}$ in $\mathcal{P}$, where:*

$$f(\mathbf{p}) = \begin{cases} 1, & \sigma \cdot \mathbf{n} > 0 \textit{ and } \mathbf{p} \textit{ is reachable from a source}, \\ 0, & \textit{otherwise}. \end{cases}$$

Point $\mathbf{p}$ in Definition 1 is the point in a sink edge at which normal vector $\mathbf{n}$ is computed. Function f ensures that only the useful flow to route agents from source to sink is considered in the maximization. Function f will lead to a max flow insensitive to the outgoing angles at sink edges, and to the possibility that parts of the flow encode arbitrary directions not useful to routing agents from source to sink.

The computational methods proposed in this paper are based on GPU-based rasterization techniques that will generate a max flow according to Definition 1, but will represent it in a discretized form, in a frame buffer grid in the GPU. The result will be a *max flow map*, as defined below.

Definition 2 (Max Flow Map.) *A max flow map for $\mathcal{P}$ is a discrete vector field $v : G \to \mathbb{R}^2$, where G is a 2D grid covering $\mathcal{P}$, and:*

$$v(\mathbf{x}) = \sigma(\mathbf{x}), \forall \mathbf{x} : |\sigma(\mathbf{x})| = 1, \textit{ where } \sigma \textit{ is a max flow in } \mathcal{P}.$$

Definition 2 captures the types of flow that our computational method produces. Basically a max flow map represents a max flow and also allows for additional directions to exist in parts of the environment that are not useful to route agents from source to sink. In both cases the flow value, or the *capacity* of the flow, for routing agents from source to sink is the same. The flow value can also be determined as the length of the polygonal min cut of the environment [11], which captures the narrowest space constraining the flow capacity. See Figure 2.

Our flow map generation method requires the computation of Shortest Path Maps (SPMs) to accumulate the values of a max flow. Next we define SPMs and we provide a self-contained description of our SPM construction method in Appendix A. A complete exposition of the method including additional details and extensions is available in our previous work [6].

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 points along the shortest collision-free path to the closest source. An SPM therefore encodes shortest paths from *all* points in a given planar environment to the closest point in a source. For any given point $\mathbf{x}$, its shortest path $\pi(\mathbf{x})$ to the closest source is reconstructed by retrieving parent points along $\pi(\mathbf{x})$ until a source is reached. A "source" here refers to a source of the SPM and is not related to a source edge of a max flow.

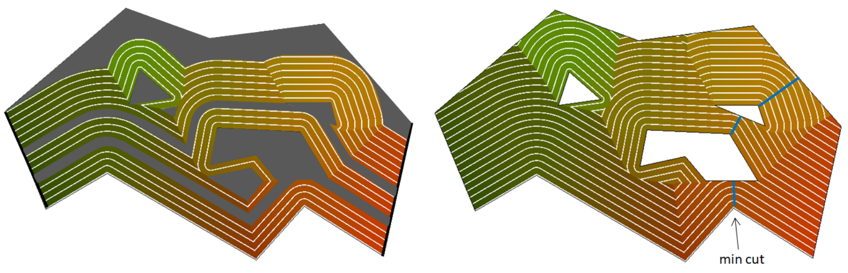


Fig. 2: Left: this max flow routes agents from source to sink edges and is illustrated with flow lines following the flow directions. The non-useful parts of the environment have directions with magnitude 0 and are shown as gray regions. Right: our equivalent max flow map includes non-null directions for the non-useful parts of the flow. While in this work our flow maps present directions as they are generated by our computational method, it would also be possible to design directions for the non-useful portions that lead agents to a useful portion of the flow. The *min cut* of the environment is represented by the 3 illustrated segments. The sum of their lengths is equal to the max flow value.

Let source points and source segments be defined inside polygonal domain $\mathcal{P}$, which also contains the set of polygonal obstacles $\mathcal{O}$. Our SPM representation encodes lengths and parent points of shortest paths, and can be defined as follows.

Definition 3 (Shortest Path Map.) *A shortest path map (SPM) in $\mathcal{P}$ is a grid-based representation $s = (s_d, s_p)$, $s_d : G \to \mathbb{R}$, $s_p : G \to \mathbb{R}^2$, where G is a 2D grid covering $\mathcal{P}$, $s_d(\mathbf{x})$ = length of $\pi(\mathbf{x})$, and $s_p(\mathbf{x})$ = parent vertex of $\mathbf{x}$ along $\pi(\mathbf{x})$. If $\mathbf{x}$ is a non-reachable point $(s_d(\mathbf{x}), s_p(\mathbf{x})) = (-1, \mathbf{x})$.*

An SPM therefore encodes, for each reachable point in $\mathcal{P}-\mathcal{O}$, 1) its geodesic distance to the closest point in a source, and 2) the next "parent point" to reconstruct the shortest path to the closest point in a source, which is always an obstacle vertex or a point in a source. Fig. 3 shows an example SPM computed for a single segment source at the bottom of the environment.

Our max flow maps are obtained by composing SPMs computed for source polygonal lines from $\mathcal{P}$ and $\mathcal{O}$. This process will be described in Section 4. However, obtaining a flow map only partially solves the navigation problem of routing multiple agents to traverse $\mathcal{P}$. After a flow map is obtained, we still need to determine where to direct agents to enter P_{src}, a process we address by determining lanes in the map. In addition, we are also interested in minimizing the length of the lanes such that the overall travel time is reduced when agents follow the lanes. Our overall approach is illustrated in Figure 4.

While the process illustrated in Figure 4 represents a complete methodology for routing agents using max flows, an alternative max flow map construction method is needed in order to guarantee that the final system of lanes

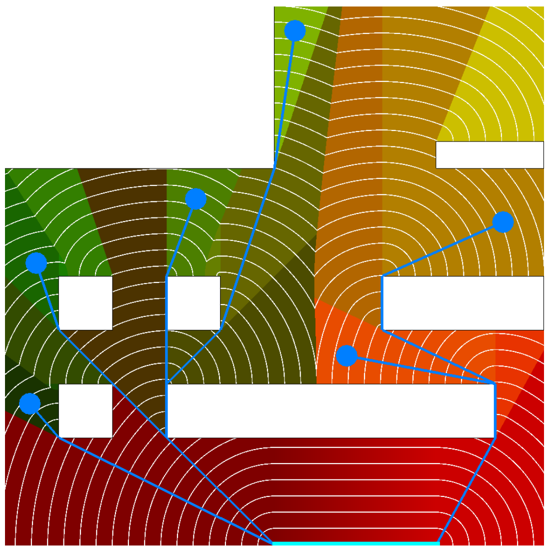


Fig. 3: Shortest Path Map (SPM) example. Contour lines represent points equidistant to the SPM's source segment (highlighted bottom segment. Discs represent agents whose polygonal shortest paths are also shown. Each region of the SPM, denoted with a same color, shares the same vertex to be taken when reconstructing a shortest path to the source segment.

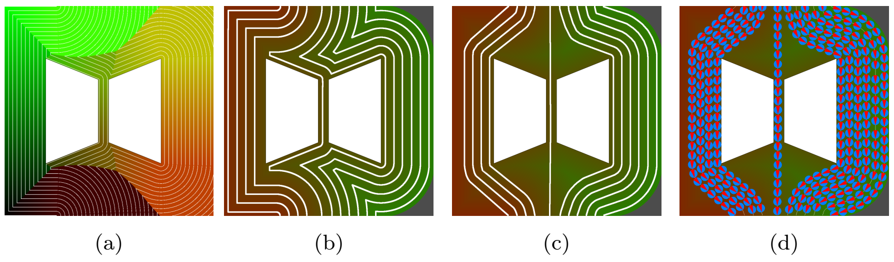


Fig. 4: Overview of the main steps of our overall approach. (a) Max flow map of the input environment. (b) Lanes extracted from the flow map. (c) Optimized lanes. (d) Using lanes to guide agents from source to sink.

utilizes the optimal maximum capacity of the environment. This alternative method is needed because the max flow map encodes a flow without observing clearance constraints. The capacity of the max flow can therefore be only guaranteed to be optimal for particles of infinitesimal size. If a max flow uses corridors in the environment with less clearance than the clearance required by an agent, that portion of the flow will not be useful for that particular agent.

We therefore introduce in this paper (in Section 5) an alternative method for generating a max flow map that takes into account the size of agents.

The approach is to contract the corridors of the environment such that the *minimum width* of the flow passing by each corridor becomes a multiple of the agent size. In this way the generated flow will allow agents to fully utilize the maximum capacity of the environment. We define this clearance-based max flow map below.

Definition 4 (CLEARANCE-BASED MAX FLOW MAP.) *A clearance-based max flow map for paths with clearance c in $\mathcal{P}$ is a max flow map $v_c : G \to \mathbb{R}^2$, where $\forall \mathbf{x} : |\sigma(\mathbf{x})| = 1$, the minimum width of the flow passing by $\mathbf{x}$ is a multiple of $2c$.*

In the above definition the minimum width of the flow passing by $\mathbf{x}$ is restricted to be a multiple of $2c$, such that the generated flow can always be fully utilized by paths of clearance c, and the restriction is limited to the flow lines that go from P_{src} to P_{snk}. If a disc agent has radius r, the path clearance needed is r, and the width of a lane is $2r$. The construction of clearance-based max flow maps is presented in Section 5. An improved lane determination method is also presented based on points evenly spaced along segments used to determine the flow width, which are called *gates*.

Overall, our presented methods are able to produce lanes among obstacles that achieve maximum flow, minimize total length, and can ensure a given clearance r.

4 Computing Max Flow Maps

We compute max flow maps following the approach described by Mitchell [11] which is based on applying Shortest Path Maps (SPMs) [13] to accumulate distances across the environment. The distance accumulation however requires to compute the SPM for the special case of considering $0/1/\infty$-cost regions. This 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) [7]. Figure 5 illustrates the difference between a traditional shortest path considering an obstacle of ∞ cost, and a shortest path considering a 0-cost obstacle.

Given the polygonal domain $\mathcal{P}$, its obstacles $\mathcal{O}$, source edges P_{src} and sink edges P_{snk}, the max flow map from P_{src} to P_{snk} will be obtained by computing the SPM with source as P_{bot} or P_{top}, and considering the obstacles to be 0-cost regions. We will denote the target SPM considering obstacles to have cost 0 as the SPM^0.

SPM^0 will accumulate distances from one boundary of the domain to the other without considering distances across obstacles. The distances that are accumulated will only encode the width of the free corridors in the environment, which will specify how much flow can pass by each corridor. The vector field defining the max flow map will then consist of the vectors orthogonal to the isolines of the obtained SPM^0.

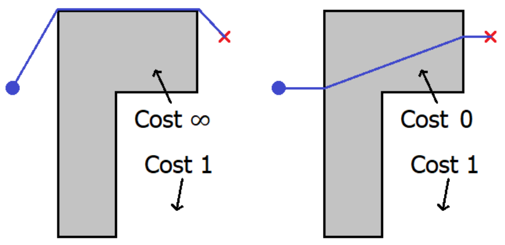


Fig. 5: Shortest pats for obstacles of infinite (left) and zero (right) cost. When an obstacle has zero cost it means that the portion of the path passing through the obstacle does not add any amount to the total cost of the path.

In order to compute SPM^0 we will apply our SPM method for regular regions multiple times, updating distances at each stage according to information obtained by first building the so-called *critical graph* of the environment.

4.1 Critical Graph

The critical graph of a polygonal domain captures key visibility information in the environment [11, 7]. Our critical graph is comprised of the shortest line segments connecting every pair of obstacles, every pair of obstacle and boundary, and P_{bot} and P_{top}, such that each segment does not cross an obstacle. In our representation these segments become the edges of the critical graph, and the obstacles become the nodes. The source and sink edges do not need to be considered for the purpose of computing SPM^0.

Figure 6 illustrates all shortest segments that are considered in order to identify the shortest ones composing our critical graph.

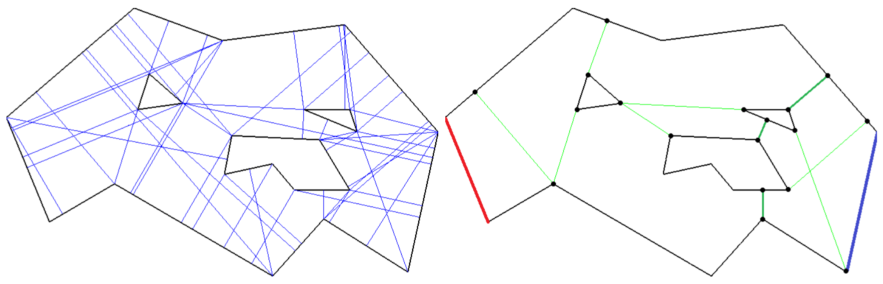


Fig. 6: Segments considered (left) in order to identify the shortest segments connecting pairs of obstacles and boundaries that compose our critical graph (right).

The critical graph encodes the width of all the corridors in the environment, and as well the pairs of obstacles and boundaries that delimit the narrowest parts of corridors. This information will be used to compute SPM^0.

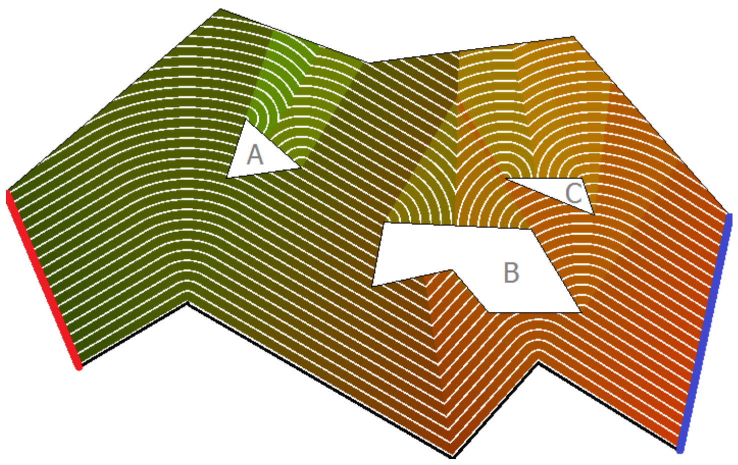
4.2 Main Algorithm

Once the critical graph of the environment is available we start by computing the SPM for segment sources which are either P_{bot} or P_{top}. In this section we choose P_{bot} as the starting polygonal line source. The process consists of the steps described below, which use Figure 7 as reference:

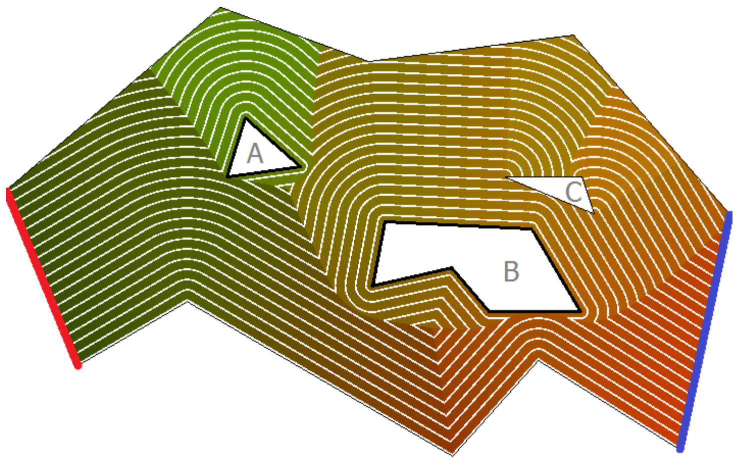
1. First, the segments of P_{bot} are set to be the initial SPM segment sources and the SPM of the scene is computed. The result is shown in Figure 7a.
2. For each obstacle O_i in the domain, the set E_i of the edges of the critical graph that connect to O_i is determined. Let $\mathbf{p}_{i_j}$ be the points that the edges in E_i connect to O_i. These are the narrowest corridor points in O_i. At each point $\mathbf{p}_{i_j}$ the accumulated distance $s_d(\mathbf{p}_{i_j})$ from the current SPM generation can be obtained from the SPM buffer. Considering all the edges in E_i, let d_i be the smallest $s_d(\mathbf{p}_{i_j})$. Each value $s_d(\mathbf{p}_{i_j})$ is compared against d_i. If d_i is not found to be smaller than any $s_d(\mathbf{p}_{i_j})$, then no shorter path to O_i was found. If this is true for every O_i, the algorithm stops and SPM^0 has been obtained. Otherwise, proceed to the next step.
3. Here d_i has a smaller distance than some $s_d(\mathbf{p}_{i_j})$, because the 0-cost of the obstacle has not been considered. The boundary of O_i is included in the list of line segments to be used as segment sources for the SPM construction of the next iteration, with the modification that d_i is used as the initial distance for the segment sources generated from O_i. The same is performed for every other obstacle O_k which is found to have a smaller d_k and is thus contributing to the new set of segment sources. Once all segment sources are identified, a new SPM is generated. However, the new values $s_d(\mathbf{p}_{i_j})$ generated will only go to the current buffer if they represent smaller distances than the values already in the buffer. Figure 7b shows the result obtained after the second SPM execution in the illustrated environment.
4. Go to step 2.

The iterations will stop when no more updates are needed. At this point SPM^0 will be obtained. Figure 7c shows the final result for the illustrated environment.

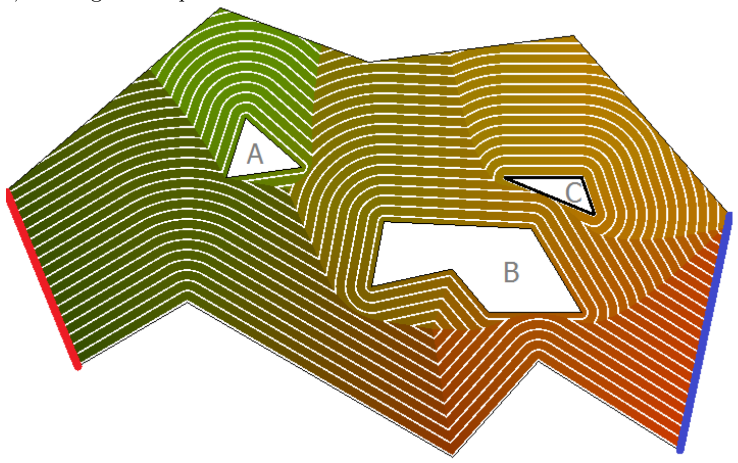
Once the process above is completed each direction of the max flow map is set to be the unit vector orthogonal to the final distance field, i.e., the max flow map vector field will store vectors parallel to the white isolines of the SPM^0 shown in Figure 7c.



(a) SPM generated with lines sources taken from P_{bot}. The red line is the source and the blue line is the sink.



(b) Obstacles A and B have shortest distances than the ones accumulated by the previous SPM, and a new iteration was performed with the boundaries of A and B as sources, altering the map.



(c) Obstacle C had a shortest distance, through obstacle B, and generated one additional iteration, finalizing the max flow map.

Fig. 7: Example steps to generate a max flow map.

4.3 Lane Extraction from a Max Flow Map

The max flow map as computed in this section can be directly used to route agents to traverse the environment. However, depending where each agent enters a source edge, it may arrive or not at the sink edge, and it is also useful to know how many agents can be deployed at the same time without creating collisions between agents. It is therefore useful to extract lanes from the flow so that agents can quickly select a free lane to use.

Lanes are represented with paths originating at P_{src} and following the flow field until reaching P_{snk}. By following lanes agents will move towards the sink in an orderly fashion without any bottlenecks. Agents always have a certain size, and although the max flow map described in this section does not consider agent size, lanes can still be determined with the needed clearance so that agents following lanes will not collide with obstacles or with other agents in adjacent lanes. We consider that each agent is represented by a circle of radius r and a lane determination process for clearance value r is therefore necessary.

We employ a simple lane determination procedure that is implemented as follows. We take points along P_{src} from an extreme endpoint towards the other extreme endpoint in order to determine candidate starting points for lanes. If the current candidate lane is found to be invalid, due lack of clearance or not reaching P_{snk}, 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, such that we consider lanes starting at every center of a cell in our grid representation G of the flow map. 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 aligned with each other, we do not need to check for collisions with previously-accepted valid lanes.

The lanes that are produced simply follow the flow and they can often display unnecessary turns in the environment. Section 6 describes a length optimization procedure that can greatly improve the overall system of lanes that is obtained. Although the lane determination procedure described in this section already incorporates clearance, the underlying flow may pass by corridors that are too narrow for a lane to pass. This may lead to a sub-optimal number of lanes generated. In order to achieve the maximum number of possible lanes with a given clearance, it is possible to generate the flow already taking into account the clearance value. This leads to a clearance-based max flow map, as described in the next section.

5 Clearance-Based Max Flow Maps

Given a pair of disjoint obstacles, it is possible to capture the narrowest passage between them with the shortest segment that connects them without crossing any obstacles. We will refer to this segment as the *gate* of the corridor between

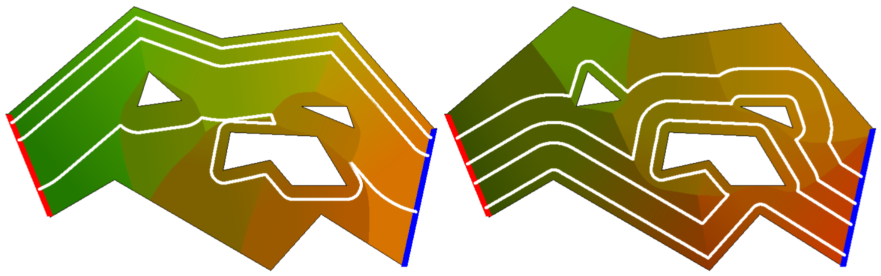


Fig. 8: Left: because of how the clearance-insensitive flow winds up in this example, a lane with clearance that simply follows it might close off a space in a way that a corridor is blocked and the maximum number of possible lanes in the environment is not extracted. Right: In this environment, when the max flow map is generated from the bottom boundary, the maximum number of lanes is correctly extracted. To ensure that generated flows always generate the maximum number of lanes the desired clearance is taken into account during the flow construction.

the two obstacles. Such segments will be directly available from the critical graph of the input environment (Figure 6-right).

The length of a gate determines the maximum number of lanes that can pass between those two obstacles. Most of the time a gate will not exactly accommodate a whole number of lanes, but rather will have some “leftover space” when a given number of lanes cross the gate. For a lane candidate to be considered valid, all gates it crosses must have enough clearance for it. However, in some cases the leftover space may be still used by the flow instead of using other corridors that might still have space for lanes to pass. This means that a clearance-insensitive max flow may flow through unpassable narrow corridors instead of larger ones. See Figure 8 for an illustration of the problem.

In order to address such situations we need to eliminate the leftover space of all corridors in the environment by forcing all gates to only fit a whole number of lanes. In this way we prevent the creation of flow portions that cannot fit a lane, and obtain the correct maximum capacity of the environment when considering lanes with clearance. The approach of contracting edges of the critical graph has been already suggested for solving *integer* flows, which addresses the equivalent case of routing wires that need to be spaced by 1 unit. The implementation of this approach in our max flow map methodology requires that we alter the boundary of the obstacles in a way equivalent to contracting gates.

First, by using the critical graph we are able to determine all the gates in the environment between obstacles and $\mathcal{P}$. We then contract every gate such that its new length is equal to a whole number of lanes based on the needed agent clearance. To do this we pick one of the endpoints of the gates, point $\mathbf{p}$, and displaced it along the gate, in the direction of the gate mid-point, by distance $l = l_g \mod 2r$, where l is the leftover space, l_g is the original gate length, and r is the radius of the agent. In doing so we obtain the contracted point $\mathbf{p}_c$. We then add the original point $\mathbf{p}$ as a vertex of the obstacle along its boundary, in case it is not already a vertex, and replace its coordinates with $\mathbf{p}_c$. In this way we locally enlarge the obstacle only enough such that the new gate will fit the same number of lanes as before, but exactly. See Figure 9.

The eliminated leftover space will force any remaining flow to be generated in different corridors. An alternative method for gate contraction would be to displace each gate endpoint by $l/2$, leading to a corrected flow with space $l/2$ at both sides of the corridor.

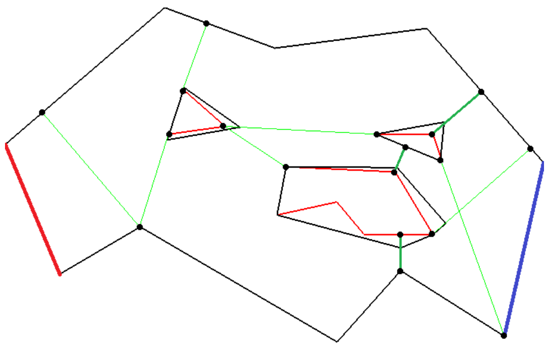


Fig. 9: The *gates*, or shortest segments, between the boundary and obstacles, and between pairs of obstacles, are contracted by displacing vertices of the obstacles such that corridors will fit an exact number of lanes. Here, as an additional optimization, the convex hull of the obstacles is adjusted. Because gate contraction is only important to distribute flows in corridors, there is no need to contract the gate between P_{bot} and P_{top}.

5.1 Improved Lane Extraction using the Min Cut

Given that the gates of the critical graph for a clearance-based max flow map are contracted, an improved lane determination method can be generated based on the min cut of the environment.

The min-cut of the environment can be found with a Dijkstra search on the critical graph. We consider the critical graph where each obstacle or boundary is a node, and the edges are the gates connecting pairs of nodes. We then consider P_{bot} (or P_{top}) to be the graph's initial node for the Dijkstra search, and P_{top} (or P_{bot}) to be the goal node. The result of the search will be the min cut, which consists of the collection of gates capturing the narrowest length to cross the environment from source to sink. Its length also determines the maximum number of lanes that can travel from source to sink.

Once the min cut of the environment is determined, we simply place lanes in the contracted gates of the min cut, with starting points evenly spaced along the contracted gates, with $2r$ space between them and with r space to obstacles or boundaries. We then follow the flow both ways from each starting point, to P_{src} and P_{snk}, in order to create the initial set of lanes. Since all gates can now fit a whole number of lanes, there is no longer a need to check if a lane has sufficient clearance. Note that because the final number of valid lanes depends also on the length of P_{src} and P_{snk}, lanes still have to be tested to reach them before being accepted, which is a trivial check.

This improved method can also be employed with the clearance-insensitive flow maps, however, the full validity tests will still have to be executed for each candidate lane.

Figure 12 shows examples where the maximum number of lanes were achieved by the clearance-based flow maps while some lanes were missed when using clearance-insensitive flow maps.

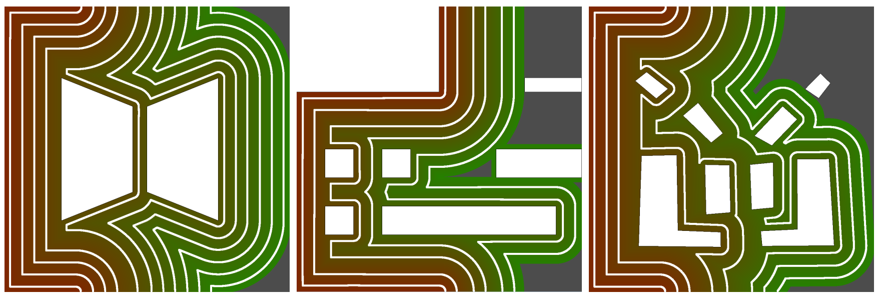
6 Length Optimization of Flow Lanes

Utilizing the max flow maps described in the previous sections and assigning agents to the extracted lanes provides an effective methodology to route 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. A post-processing optimization process to reduce this inefficiency can then be applied.

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 $\overline{p_1 p_2}$ is a valid "shortcut." This is only true if $\overline{p_1 p_2}$: does not intersect with any other occupied lane or obstacle segment, keeps its minimum distance to all obstacles and $\mathcal{P}$ as at least r, and keeps its distance 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 initiated the generation of the max flow map, and therefore the last lane tends to have the most open space to be optimized. 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 10.


(a) original lanes

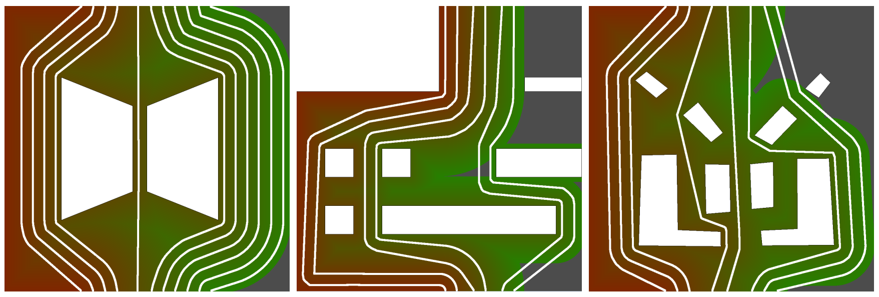

(b) optimized lanes

Fig. 10: Scenarios 1 (left), 2 (middle), and 3 (right).

7 Results and Discussion

Our methods have been evaluated by producing several max flow maps and lane systems for a variety of environments. We have also produced simulation examples comparing the benefits of using our max flow trajectories versus having agents simply following their shortest paths.

7.1 Efficiency for Flowing Agents

In order to illustrate the benefits of using our max flow trajectories we have produced multi-agent simulations employing our generated lanes. One simulation is illustrated in Figure 11 and the results obtained are summarized in Table 1. 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$_{snk}$

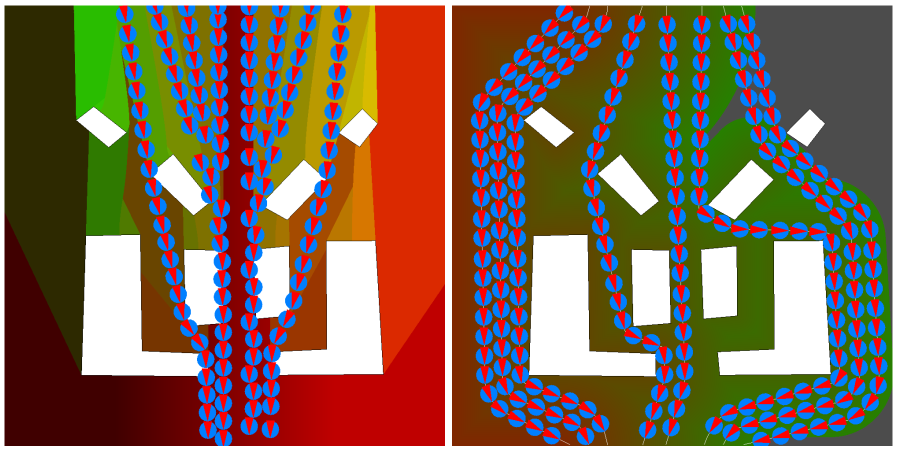


Fig. 11: Snapshots of simulations on scenario 3. Despite both having the same amount of space and 8 lanes to start with, the SPM$_{snk}$ only permits 4 agents to reach the exit at a time, while the max flow map permits all 8 to do so.

	Scenario 1				**Scenario 2**				**Scenario 3**			
	min	**max**	**avg**	**n**	**min**	**max**	**avg**	**n**	**min**	**max**	**avg**	**n**
S:	1.97	2.02	1.99	453	2.24	2.28	2.26	214	1.97	2.03	2.00	441
L:	2.57	3.92	3.06	1053	3.30	4.39	3.89	553	2.74	4.31	3.20	766
O:	1.97	2.52	2.39	1165	2.65	3.61	3.09	611	1.98	2.89	2.40	843

Table 1: The left-most column indicates the used method. S: shortest paths to sink using SPM$_{snk}$. L: lanes from the max flow map. O: optimized lanes from the max flow map. The simulations had the agents continuously spawn at the source whenever there was space for them, and then the agents moved towards the sink according to the used method. The simulated period was 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. 10.

simply encodes shortest paths and does not include any flow information. The first row in Table 1 presents the results.

2) The second type uses our max flow map of the environment, built starting the underlying generation from the P_{bot} side. The flow map provides directions to agents placed anywhere in the covered regions of the environment but the retrieved lanes are used to guide the agents. The lanes are optimal with respect to the flow capacity but their lengths can be further optimized. The second row in Table 1 presents the results.

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

achieving the max flow of the environment. The results are presented in the third row of Table 1.

In each environment type we repeatedly spawn agents at the source edge whenever there is space for them, as the agents use either the SPM$_{snk}$, lanes, or optimized lanes, to navigate towards the sink. Whenever an agent reaches the sink, it is removed from the environment. The example in Figure 11 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 1. 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 led to 2 or sometimes close to 3 times as many agents reach their destination. Also, agents using the max flow map lanes did not require collision avoidance behavior. In these examples both the used max flow maps and the respective clearance-based max flow maps would lead to the same number of lanes. While lanes were slightly different, and with slightly different lengths, no significant difference on the reported values would be expected.

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 number of agents that can reach the sink polygonal line at the same time without bottlenecks.

7.2 Additional Results

Several additional results are presented in Figures 12, 13, and 14.

Figures 12 and 13 show the benefits of clearance-based flow maps. The top rows of images in Figures 12 and 13 show clearance-insensitive flow maps which lead to lanes being missed. The maps on the bottom rows are where we apply clearance-based flow maps in order to find the maximum number of lanes.

Figure 14 presents the results of our methods on two additional environments with higher number of obstacles. The lanes generated on the left images are unoptimized, whereas the right images shows the same lanes after length optimization.

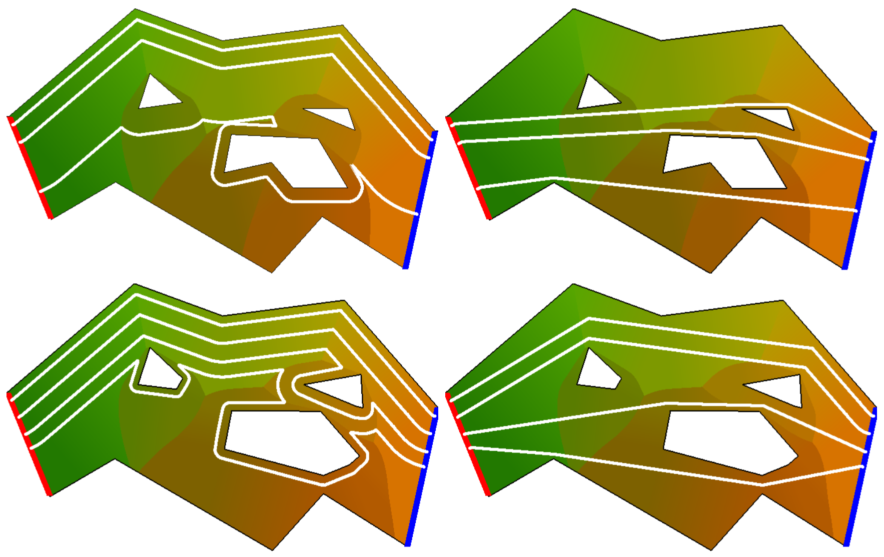


Fig. 12: The top row shows a clearance-insensitive max flow map missing one lane compared to the clearance-based max flow map in the bottom row. The right column shows the same lanes after a length optimization process.

7.3 Limitations and Future Work

Our current lane determination algorithms evaluate lanes by considering positions at the source or min cut edges without considering any additional global information from the environment. This may lead to a poor choice of lanes in some specific situations. For example, the lanes shown in Figure 15 achieve the maximum possible flow but miss the shortest path from source to sink. This situation might be addressed by first assigning lanes in the gates containing a shortest path from source to sink, and then proceeding to the remaining gates of the min cut. Another possible improvement is to allow updating the endpoints of each lane to alternative positions, in P_{src} or P_{snk}, in order to further optimize their length. We have also noticed cases where the generated flow wraps around it and comes to intersect again the source segment; however, our lane determination procedures ensure that no lanes are generated in these areas.

One promising direction for future work is to address crossing flows. For instance, groups of agents may be defined as each group having its own goal sink, and one specific flow map for each group can be then computed. Later, it might be enough to deploy agents with the right timing such that they do not collide with each other when following their on flow paths. Reactive behaviors can also be used to avoid collisions at flow crossings. Such possible approaches, among others, would allow the proposed methods to be used for agents with different goal locations.

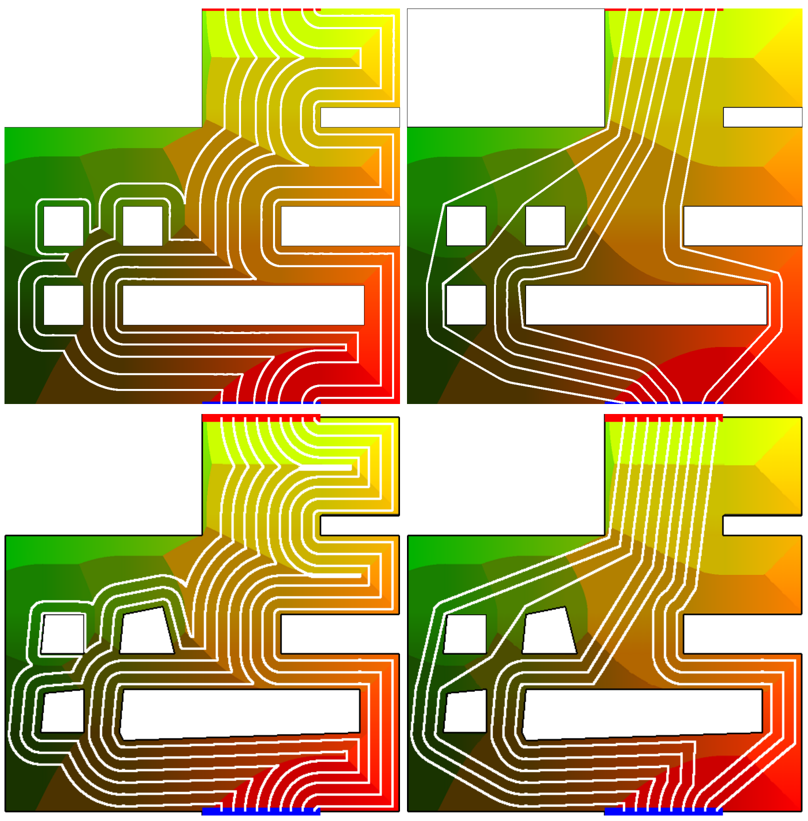


Fig. 13: In the top row, the clearance-insensitive max flow map misses two lanes compared to the clearance-based max flow map in the bottom row. The right column shows the same lanes after a length optimization process.

It is also possible to address max flow maps with multiple disconnected sources and sinks. In this case there are disconnected edges in P_{src} and/or P_{snk} and there is no longer just a pair of segments on the boundary that can be divided into P_{top} and P_{bot}, but rather many segments. By applying every boundary segment that is not part of P_{src} or P_{snk} to be the starting sources of the underlying generation of SPM^0, we can compute a map such as the one illustrated in Fig. 16, which produces a flow routing multiple entrances to multiple exits. This approach however has the limitation that any source may be routed to any sink.

8 Conclusion

We have introduced in this paper new techniques to compute max flow maps capturing the maximum flow capacity of given generic polygonal domains.

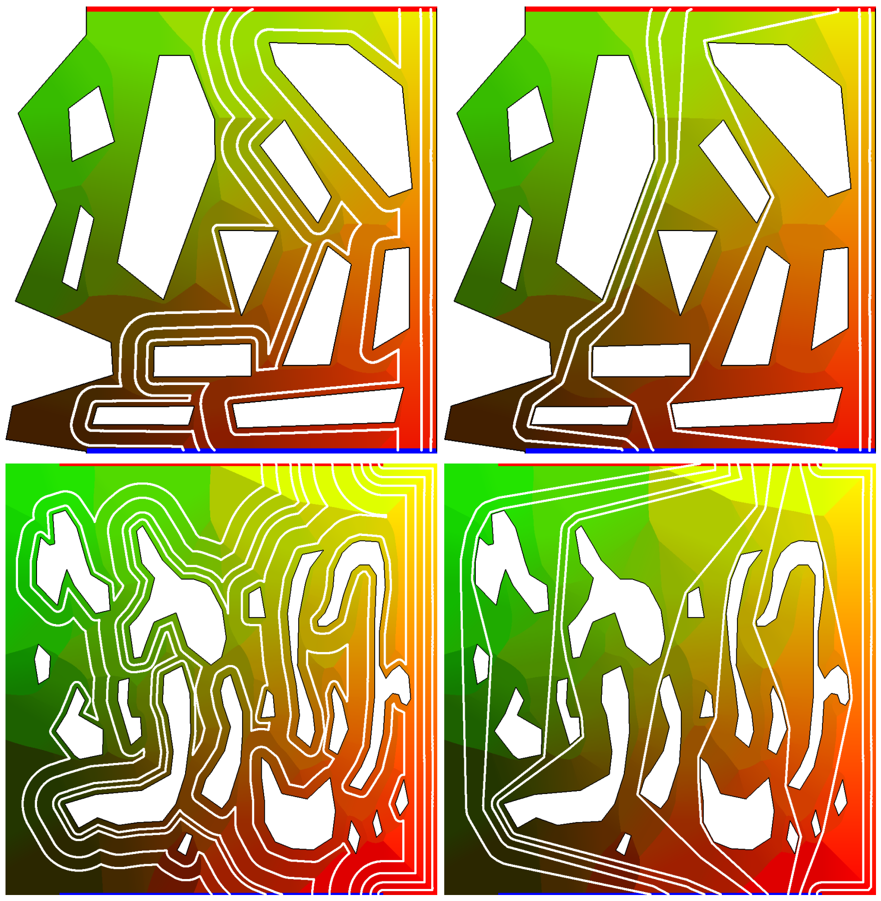


Fig. 14: Examples generating the maximum number of lanes before (left) and after (right) length optimization.

The proposed methods are able to determine bottleneck-free lanes that are able to optimally guide agents to reach a destination exit of the environment. Optimality is addressed with respect to the maximum flow of agents across the environment. The presented simulations demonstrate that our approach can dramatically increase the number of agents that successfully navigate towards the goal exit of the environment in a given time frame.

The proposed approach introduces a new methodology for taking into account continuous flows in polygonal domains, and exposes several promising directions for future work, opening new research avenues in flow-based agent navigation.

Acknowledgements 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 repre-

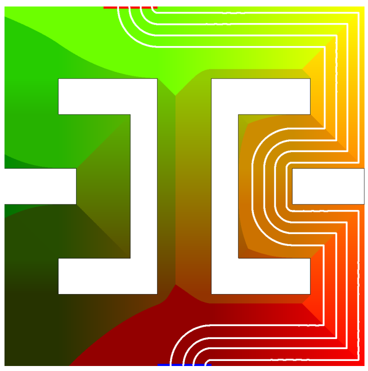


Fig. 15: In this environment lanes are inefficient in terms of length whether the map is generated from both P_{bot} or P_{top}.

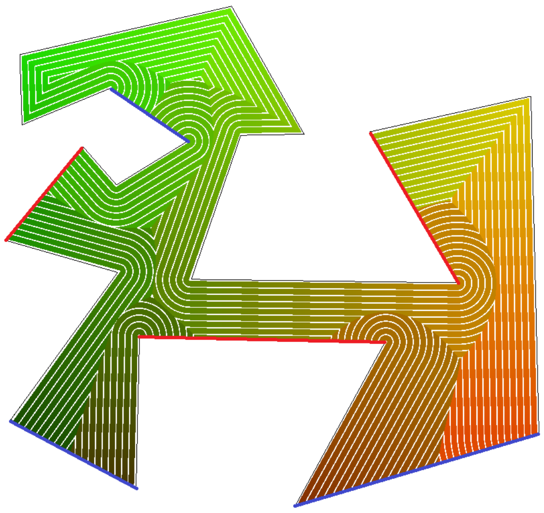


Fig. 16: A flow map with multiple sources and sinks. The red lines are source segments belonging to P_{src} and the blue lines are sink segments belonging to P_{snk}. Every segment on the boundary inbetween them is used in the SPM generation process and thus the map is created such that agents from any source may travel to any sink, preventing crossing lanes.

senting 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. Barnett A, Shum HPH, Komura T (2016) Coordinated crowd simulation with topological scene analysis. Computer Graphics Forum 35(6):120–132
2. Berczi K, Kobayashi Y (2017) The Directed Disjoint Shortest Paths Problem. In: Pruhs K, Sohler C (eds) 25th Annual European Symposium on Algorithms (ESA 2017), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Leibniz International Proceedings in Informatics (LIPIcs), vol 87, pp 13:1–13:13
3. Camporesi C, Kallmann M (2014) Computing shortest path maps with GPU shaders. In: Proceedings of the Seventh International Conference on Motion in Games, ACM, New York, NY, USA, MIG ’14, pp 97–102
4. Dobson A, Solovey K, Shome R, Halperin D, Bekris KE (2017) 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, pp 120–127
5. Farias R, Kallmann M (2018) GPU-based max flow maps in the plane. In: Proceedings of Robotics: Science and Systems (RSS)
6. Farias R, Kallmann M (2019) Optimal path maps on the GPU. IEEE Transactions on Visualization and Computer Graphics
7. Gewali L, Meng A, Mitchell JS, Ntafos S (1988) Path planning in $0/1/\infty$ weighted regions with applications. In: Proceedings of the Fourth Annual Symposium on Computational Geometry, ACM, New York, NY, USA, SCG ’88, pp 266–278
8. Karamouzas I, Geraerts R, van der Stappen AF (2012) Space-time group motion planning. In: Workshop on the Algorithmic Foundations of Robotics
9. Ma H, Koenig S (2016) Optimal target assignment and path finding for teams of agents. In: Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, AAMAS ’16, pp 1144–1152
10. Ma H, Koenig S (2017) AI buzzwords explained: multi-agent path finding (MAPF). AI Matters 3(3):15–19
11. Mitchell JSB (1988) On maximum flows in polyhedral domains. In: Proceedings of the Fourth Annual Symposium on Computational Geometry, ACM, New York, NY, USA, SCG ’88, pp 341–351
12. Mitchell JSB (1990) On maximum flows in polyhedral domains. Journal of Computer and System Sciences (40):88–123
13. Mitchell JSB (1991) A new algorithm for shortest paths among obstacles in the plane. Annals of Mathematics and Artificial Intelligence 3(1):83–105
14. Sharon G, Stern R, Felner A, Sturtevant NR (2015) Conflict-based search for optimal multi-agent pathfinding. Artif Intell 219(C):40–66
15. Solovey K, Yu J, Zamir O, Halperin D (2015) Motion planning for unlabeled discs with optimality guarantees. In: Robotics: Science and Systems

16. Strang G (1983) Maximal flow through a domain. Mathematical Programming 26(2):123–143
17. Surynek P (2010) An optimization variant of multi-robot path planning is intractable. In: AAAI
18. Svancara J, Surynek P (2017) New flow-based heuristic for search algorithms solving multi-agent path finding. In: ICAART (2), SciTePress, pp 451–458
19. Van Den Berg J, Guy S, Lin M, Manocha D (2011) Reciprocal n-body collision avoidance. In: Robotics Research - The 14th International Symposium ISRR, no. STAR in Springer Tracts in Advanced Robotics, pp 3–19
20. Yu J, LaValle SM (2013) Multi-agent path planning and network flow. In: Algorithmic Foundations of Robotics X, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 157–173
21. Yu J, LaValle SM (2013) Planning optimal paths for multiple robots on graphs. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp 3612–3617
22. Yu J, LaValle SM (2013) Structure and intractability of optimal multi-robot path planning on graphs. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, AAAI Press, AAAI'13, pp 1443–1449

Appendix

A Computation of SPMs Based on GPU Rasterization

This appendix describes the method employed in this work for computing SPMs. The full description of the method is available in our previous work [6], which also includes several extensions. An earlier version of the approach, limited to point sources, was introduced by Camporesi et al. [3].

A.1 SPMs for Point Sources

We first describe the approach considering only point sources. The approach considers the input environment to be described in plane $z = 0$ of a 3D space, and then rasterizes "clipped cones" with apices placed at specific depths below the 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 (its *geodesic distance*). When a cone is rasterized, meaning it is converted to the pixels which it occupies in a GPU buffer (placed at $z = 0$ plane), the depth values of the affected pixels increase proportionally to their Euclidean distances to the apex, and will thus contain the geodesic distances to the closest point source. Furthermore, in cases of overlap, the GPU's depth test will only allow the smallest value to remain in the frame buffer. Fig. 17 illustrates the steps of the process and the concept of clipped cones.

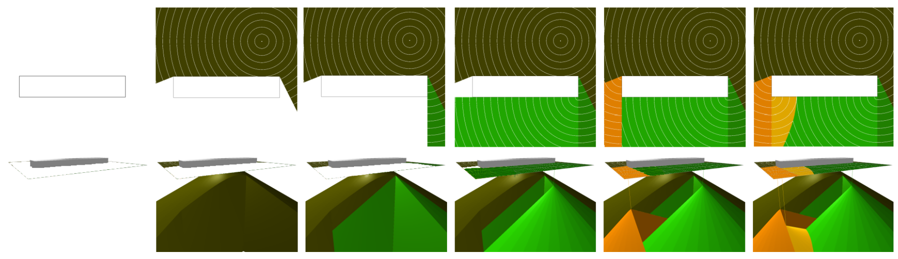


Fig. 17: The steps of the SPM construction process for an environment with one source point and an obstacle with four vertices [6]. The cones are "clipped" by the visibility of the point that generated it. The top row shows the GPU buffer at each step. The bottom row is a 3D view of the underlying cones for illustration purposes.

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." A generator point will be a parent point to the previous vertex along a shortest path reaching the generator.

The GPU framebuffer stores the following information for each pixel: 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. This representation encodes all the information described in Definition 3.

The SPM generation process consists of choosing a generator from the array and rasterizing its clipped cone. Each generator is processed once.

First we determine which generator will rasterize cone next by choosing the one with the smallest distance to a source, among the ones that can become generators. Source points will be the first ones to be picked because their distance to sources 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 cone. The cone apex is placed under the generator at a depth equal to the geodesic distance of the generator. When the cone is rasterized, only the parts of the scene which have direct line-of-sight to the generator are to be affected, and therefore the cone is "clipped" against the region visible from the generator. This visibility determination is part of the construction of the clipped cones, and is done before the actual rasterization by determining the visible set in a dedicated buffer, and using it for the clipping. The pixels that are affected by the rasterization will thus have direct line-of-sight to the generator point, so they can calculate their Euclidean distance to it and add it to the generator's geodesic 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 represent the desired SPM.

A.2 SPMs for Segment Sources

There are two main modifications needed in the basic algorithm in order to consider segment sources. First, segment sources have to be decomposed in sub-segments at *critical points* which encode where the visibility of the scene changes with respect to the segment. The other extension is to rasterize distances that come from "elongated" cones, instead of cones, when the associated generator is a segment or sub-segment instead of just points.

Let $\mathbf{l}_i$ be a source segment with n_{c_i} *critical points*, $n_{c_i} \geq 0$. A critical point defines a point where the orthogonal projection of an obstacle edge to the segment changes to a different edge of the same obstacle. Critical points are only considered when obstacle vertices can be projected to a segment through a projection line not crossing any obstacles. See Figure 18 for examples.

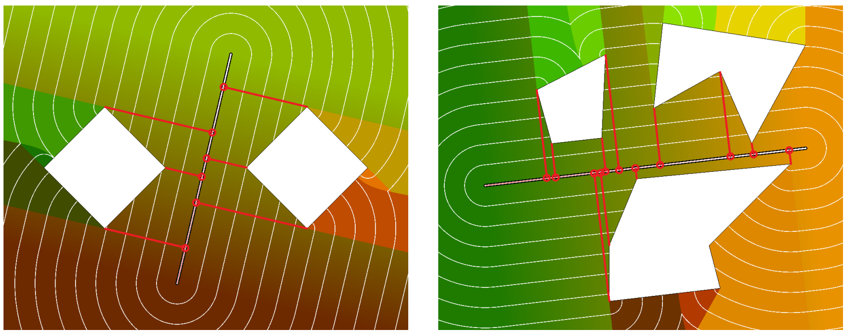


Fig. 18: The circled points on the segment sources are the critical points, which are projections of obstacle vertices.

Since critical points denote points where the visibility of the scene changes with respect to the segment, we must then consider each of the sub-segments between the endpoints and the critical points independently. We then consider that $\mathbf{l}_i$ is now composed 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 defines a sub-segment that is included in the SPM generation process as an elongated cone. The basic SPM generation algorithm then proceeds. Elongated cones emanating from segment sources may reach visible obstacle vertices, which will eventually generate new cones during the propagation process and all generated costs are correctly merged in the final SPM representation. One example SPM with a line segment source is illustrated in Figure 3.