Efficient Traffic Routing with Progress Guarantees

Stefan Blumer
ETH Zurich
blumers@ethz.ch

Manuel Eichelberger
ETH Zurich
manuelei@ethz.ch

Roger Wattenhofer
ETH Zurich
wattenhofer@ethz.ch

***Abstract*—This paper presents an efficient traffic scheduling algorithm for vehicles such as cars, trains or ships. We provide guarantees for deadlock and starvation freedom, therefore ensuring progress for each vehicle in the system. Our method tolerates vehicles which do not disappear from the traffic network once they reach their destination, but rather continue towards subsequent destinations. Therefore, vehicles can run indefinitely. We introduce the concept of *safe spots*, which are locations where a vehicle can stop without ever blocking another vehicle. Using such safe spots, we divide routes into short segments, which reduces the number of routing alternatives exponentially, thus allowing real-time traffic allocation.**

***Index Terms*—autonomous car, deadlock freedom, naval traffic, network utilization, railroad, train network, ship, starvation freedom**

I. Introduction

Many people spend a fair amount of time in traffic, for instance for commuting to work. In the US, the average car driver spends an equivalent of seven(!) 40-hour work weeks per year in traffic.[1] A considerable part of this time is wasted in traffic jams.[2] This is a nuisance, which is caused by the limited capacity of street networks and by inefficient traffic allocation and routing. Also in train networks, traffic density is a challenge. Even in Switzerland, which has one of the densest rail networks in the world, traffic is reaching the capacity limits.[3] Expanding road and rail networks is costly and uses land. Land is especially precious in urban areas, where the traffic volume is highest and traffic capacity increases are usually most profitable. Instead of, or complimentary to, expanding traffic infrastructure like streets and rails, better routing offers a cheaper way of improving the life of traffic participants.

In this paper, we propose a traffic routing system which allows for efficient scheduling with variable safety margins around vehicles with a particular focus on preventing deadlocks and starvation. In high-density real-world traffic, deadlocks do occur at some frequency and are hard to resolve.[4] Sure enough, dense traffic exists in many places in the world, and measured by this fact, traffic deadlocks are still relatively uncommon on a global scale. However, this often comes at the price of inefficient crossroads, which for instance feature long idle times between traffic crossing from different directions. In both cases, an implementation of our proposed system could improve the situation.

Besides car and train networks, which can be well modeled as graphs, traffic routing is also useful for ships. Ship navigation is often not constrained to strict lanes and therefore allows ships to move freely, which results in more possibilities for vessel allocation. Also, differently sized ships do not need the same "lane width". Several collisions of naval vessels in the year 2017 highlight that manual navigation, even with modern navigation aids, is prone to errors and can result in fatalities and million-dollar damages.[5] Better routing with sufficient, but not thriftless, safety margins and automated navigation can increase the reliability of vessel traffic and improve the traffic throughput at the same time.

Besides preventing deadlocks and guaranteeing progress for each vehicle in the system, our method also tolerates vehicles which move on infinitely long paths. Solving one of these problems alone is relatively easy: As long as vehicles want to travel from one origin to one destination, a simple shortest path search on a graph with nodes corresponding to a geographic location at a certain time guarantees deadlock freedom and progress, as long as the nodes on the path are deleted for subsequent searches for other vehicles' paths. Conversely, if deadlocks or starvation do not need to be prevented, each vehicle can start a new graph search to its next destination, independently of all other vehicles, upon reaching an intermediate destination. However, this latter method does not prevent deadlocks: For instance, if two vehicles' intermediate destinations are in the same single-lane dead end, once the first vehicle reaches the destination, both vehicles start moving towards each other and mutually block their way. Our framework prevents any such situations.

Furthermore, the introduced concept of *safe spots* allows for network partitioning, creating the possibility of local traffic scheduling by decentralized authorities.

Our abstract method for traffic routing can be adapted to different means of transport such as railroads and ships. It is not only efficient in the allocation of vehicles in a given traffic network, but also has low computational requirements, allowing it to run in real-time on a consumer grade computer.

[1]American Automobile Association: https://newsroom.aaa.com/2016/09/americans-spend-average-17600-minutes-driving-year/

[2]Reuters: https://www.reuters.com/article/us-usa-traffic-study/u-s-commuters-spend-about-42-hours-a-year-stuck-in-traffic-jams-idUSKCN0QV0A820150826

[3]Swissinfo: https://www.swissinfo.ch/eng/swiss-railways-heads-towards-its-limit/4364

[4]Many images and videos on the Internet show traffic deadlocks. Search for "traffic deadlock" for examples.

[5]Ars Technica Report: https://arstechnica.com/gadgets/2017/08/with-the-uss-mccain-collision-even-navy-tech-cant-overcome-human-shortcomings/

II. Related Work

Routing is a well researched topic in computer science. In simulations and computer games, it is also called *pathfinding*. For traffic routing, we are mainly looking at variants of the *shortest path problem* which is the special case of routing with the objective function of minimizing the cost of a path.

The most famous algorithm for solving the single-source shortest path problem is Dijkstra's algorithm [1]. Several generalizations exist, which for instance tolerate negative or probabilistic edge weights. Important for our method is the A* search algorithm, which uses heuristics to speed up searches.

Our setting is a multi-source multi-destination routing problem in which the needs of multiple vehicles, or more generally abstract *agents*, have to be accommodated. Generally, this problem is NP-hard to solve optimally [2]. Techniques finding optimal multi-agent pathfinding (MAPF) solutions are therefore slow. Many such methods are based on variants of A* [3], [4]. Others reduce the problem to NP-complete problems such as SAT [5] or Integer Linear Programming [6].

Some research on MAPF focuses on heuristic approaches [2]. So far, heuristic algorithms are capable of computing one collision-free path for each of several agents with arbitrary start and target locations. Concerning the properties of the end state, when each agent has reached its target location, to the best of our knowledge, no previous work considers whether agents can reach further destinations.

One work experimentally determines the best MAPF algorithm for naval traffic in specific bounded regions [7]. In that paper, the goal is to minimize *unsafe* paths, defined by a geographical and temporal distance threshold, which might lead to collisions. Such unsafe paths are reduced but not entirely prevented. An inherent assumption is that agents can always leave the system within finite time and there will be no deadlocks as soon as the ships are in open waters. Compared to our method, the heuristics do not guarantee deadlock freedom, but can work well in practice.

As explained in the introduction, we are not just considering multiple agents, but looking at the more difficult problem that involves agents which follow infinite paths. For this setting, we could not find any related academic work. However, we believe that this is an important branch of research, as the example with the dead end illustrates.

In the industry, traffic simulations are for instance used to predict traffic in city centers or during emergency evacuations and in video games. Pathfinding implementations for video games focus more on computational performance than physical accuracy or progress guarantees. Some games naively perform pathfinding to the next target as soon as the previous one is reached. Like this, deadlocks are not prevented and even occur frequently in some games. Other games simply allow agents to move through each other. This simplifies the simulation, since each vehicle's path can be computed independently of all others and still prevents deadlocks or starvation. However, in the real world, the physical nature of traffic has to be respected in any scheduling mechanism.

III. Method

The basic idea of our technique to find paths for multiple agents moving along infinite routes is to compute partial paths and repeatedly expand them. This is necessary, since general infinite paths cannot be computed with finite computation power and storage.

In contrast to the naive approach, in which the search is done from destination to destination, we perform searches between so-called *safe spots* to guarantee that no two agents ever end up in a deadlock situation. Unlike the destinations, safe spots are not known a priori and have to be found. In practice, paths can be expanded before an agent reaches the next safe spot. This can improve the route quality in situations with a limited number of safe spots, by omitting detours for passing through a safe spot. Nevertheless, this does not thwart the deadlock and progress guarantees provided by the safe spots.

The partial paths are computed using a pathfinding technique, such as Dijkstra's algorithm [1]. While our method does not rely on a specific pathfinding algorithm, the choice influences the computation performance.

A. Setup

Our technique works on graphs. Every state which an agent can reach is represented by a node in a graph. States may not only consist of an agent's position, but any variable relevant to the movement like velocity, rotation, fuel or time. An agent's movement restrictions determines the edges of its movement graph.

For multiple agents, we could consider their combined states and movement restrictions to build one collective graph with each node representing all agents' states and each edge corresponding to movement actions for all agents. However, such a graph grows exponentially with the number of agents in the system, so this approach is infeasible in practice. Instead, we construct individual graphs for all agents and compute movement steps sequentially. For each agent's move, the invalidated edges of the graphs of all other agents are deleted. In the end, the agents can move all at the same time on their computed paths.

The following notation is used:

- Agents $A = \{a_1, \dots, a_n\}$
- Graph nodes $V = \{v_{11}, v_{12}, \dots, v_{1m_1}, v_{21}, \dots, v_{nm_n}\}$
- Stations $S = \{S_1, \dots, S_q\}, \forall i.\ S_i \subseteq V$
- Path $P = (v_{ik}, v_{il}, \dots)$
- Route $R = (S_i, S_j, \dots)$

An agent a_i always stays on its part of the graph consisting of the nodes $\{v_{i1}, \dots, v_{im_i}\}$. Stations are sets of nodes, which allows modeling different station sizes. An agent reaches a station when arriving at any of its nodes. A path is a finite sequence of nodes which an agent visits consecutively. Similarly, routes are an *infinite* sequence of stations, with one route for each agent. Agents are indefinitely moving along their route, to any of the next station's nodes.

Definition 1 (Collision). $Collision(v_{ik}, v_{jl})$ is a symmetric boolean function, which is true if nodes v_{ik} and v_{jl}

cannot simultaneously be occupied (by agents a_i and a_j). $Collision(v_{ik}, v_{il})$ is always false.

A path P is valid if at any time $\forall v_{ik} \in P, v_{jl} \in V \setminus v_{ik}$. $Collision(v_{ik}, v_{jl}) =$ false $\vee\, v_{jl}$ is unoccupied, so no collision occurs at any time.

Definition 2 (Movement Invariant)**.** Each agent is at any time either moving towards a safe spot or staying at a safe spot.

B. Safe Spots

Definition 3. A *safe spot* is a node where an agent may stay forever, such that at any time, for every pair of other safe spots in any agent's graph, there is still a connecting path.[6]

A maximal set of safe spots can be constructed by repeatedly adding new nodes while showing the existence of alternative paths. Alternative paths must cross neither the new safe spot candidate nor any existing safe spots.

Below, let $T_i \subseteq S$ be the sequence of stations on the path of agent a_i, and K the set of already selected safe spots.

Definition 4 (Path Existence)**.** $PathExists(v_{ik}, v_{il}, K)$ denotes the existence of a path P from node v_{ik} to safe spot $v_{il} \in K$, passing through all stations $S_i \in T_i$ such that

$$\forall v_{ir} \in P, v_{js} \in K \setminus \{v_{il}\} : \neg Collision(v_{ir}, v_{js})$$

Definition 5 (Alternative Path)**.** $Alternative(v_{ik}, v_{jl}, K)$ is true iff there exists an alternative path around v_{jl} for every path segment starting at safe spot $v_{ik} \in K$:

$$Alternative(v_{ik}, v_{jl}, K) \coloneqq \forall v_{ih} \in K : \\ PathExists(v_{ik}, v_{ih}, K) \Longrightarrow PathExists(v_{ik}, v_{ih}, K \cup \{v_{jl}\})$$

Definition 6 (Potential Safe Spot)**.** $v_{jl} \notin K$ is a potential safe spot iff

$$\forall v_{ik} \in K : Alternative(v_{ik}, v_{jl}, K).$$

Every potential safe spot can be selected as a safe spot. The sequential selection of safe spots forms a strict ordering of safe spots.

Definition 7 (Safe Spot Ordering)**.** For safe spots v_{ik} and v_{jl}, we define $v_{ik} < v_{jl}$ iff v_{ik} is selected as safe spot before v_{jl}.

Theorem 8. *In a set of safe spots $K \subseteq V$, each safe spot ensures alternative paths for all previously selected safe spots.*

$$\forall v_{ik}, v_{jl} \in K : v_{ik} < v_{jl} \Longrightarrow Alternative(v_{ik}, v_{jl}, K \setminus \{v_{jl}\})$$

Proof. The claim follows directly from Definition 6. □

Whether a node can become a safe spot depends on the set of already selected safe spots. Initially, when the set of safe spots is empty, any node can become a safe spot, since no alternative paths need to be found. Depending on the sequence of safe spot choices, the maximal number of safe spots may differ. Note that according to Definition 2, an agent is only allowed to enter the traffic network if there exists at least one safe spot in its part of the graph.

[6]An inherent weak assumption is that the movement restrictions of an agent allow it to stay at the node of the movement graph for an unbounded time. This usually represents an agent standing still.

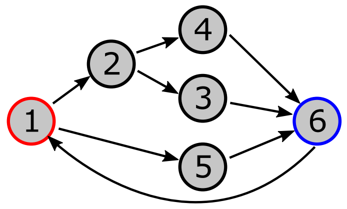

Fig. 1. Movement graph for a single agent with stations $S_1 = \{1\}$ and $S_2 = \{6\}$

Example 9. Initially, all nodes of the graph in Figure 1 are potential safe spots. If Node 3 is selected as the first safe spot, Nodes 4 and 5 are further potential safe spots. If Node 2 is selected instead, Nodes 3 and 4 are other potential safe spots.

C. Deadlocks

According to Definition 2, agents always move between safe spots. A deadlock occurs if agents cannot find a path to another safe spot, because all paths are blocked by other agents. In this section, we show that deadlocks *cannot* arise.

Definition 10 (Lock)**.** We say that (an agent in) a safe spot *locks* another one, when there is no alternative path around the former, for paths starting at the latter:

$$Lock(v_{jl}, v_{ik}, K) \coloneqq \neg Alternative(v_{ik}, v_{jl}, K) \vee \\ (\exists v_{gh} \in K \setminus \{v_{jl}\} : Lock(v_{jl}, v_{gh}, K) \wedge Lock(v_{gh}, v_{ik}, K))$$

The second part of the function makes it transitive. This captures cases in which two agents lock each other indirectly.

Definition 11 (Deadlock)**.**

$$Deadlock(v_{ik}, v_{jl}, K) \coloneqq Lock(v_{ik}, v_{jl}, K) \wedge Lock(v_{jl}, v_{ik}, K)$$

To prove deadlock freedom, we show that no two agents in a valid state according to Definition 2, can be in a deadlock.

Theorem 12. *A safe spot v_{jl} cannot lock another safe spot v_{ik} if the latter was selected as a safe spot before v_{jl}:*

$$v_{ik} < v_{jl} \implies \neg Lock(v_{jl}, v_{ik}, K)$$

Proof.

$$\neg Lock(v_{jl}, v_{ik}, K) \iff Alternative(v_{ik}, v_{jl}, K) \wedge \\ (\forall v_{gh} \in K \setminus \{v_{jl}\} : \neg Lock(v_{jl}, v_{gh}, K) \vee \neg Lock(v_{gh}, v_{ik}, K))$$

Our proof goes by induction.
Base case: An alternative path is ensured by Theorem 8.
Induction step: To prove $\forall v_{gh} \in K \setminus \{v_{jl}\} : \neg Lock(v_{jl}, v_{gh}, K) \vee \neg Lock(v_{gh}, v_{ik}, K)$, we make a case distinction:

1) Case $v_{ik} < v_{gh}$: We recursively apply Theorem 12 to get $\neg Lock(v_{gh}, v_{ik}, K)$.
2) Case $v_{gh} < v_{ik}$: Due to the total order provided by Definition 7, $v_{gh} < v_{jl}$ and we recursively apply Theorem 12 to get $\neg Lock(v_{jl}, v_{gh}, K)$.

In a finite graph, the transitive function $Lock(v_{ab}, v_{cd}, K)$ can only be formed from a finite list of non-transitive relations $Lock(v_{ab}, v_{g_1h_1}, K), Lock(v_{g_1h_1}, v_{g_2h_2}, K), \ldots, Lock(v_{g_ph_p}, v_{cd}, K)$. Otherwise, there must be some loop $Lock(v_{wx}, v_{yz}, K), Lock(v_{yz}, v_{wx}, K)$, where $Lock(v_{wx}, v_{yz}, K)$ is non-transitive and, because of the strict ordering of safe spots, $v_{wx} < v_{yz}$ holds. By Theorem 8 we have $Alternative(v_{wx}, v_{yz}, K \setminus \{v_{yz}\})$. Using Definition 10, this leads to $\neg Lock(v_{wx}, v_{yz}, K)$ which breaks the chain of non-transitive relations above. Therefore, the recursion depth of applying Theorem 12 is finite. □

Theorem 13. *Any two agents in safe spots are never in a deadlock situation:*

$$\forall v_{ik}, v_{jl} : v_{ik} < v_{jl} \implies \neg Deadlock(v_{ik}, v_{jl})$$

Proof. This follows from Theorem 12, since one of the conditions for a deadlock, as in Definition 11, is not satisfied. □

D. Progress

Moving from safe spot to safe spot does not necessarily mean that an agent advances on its route. However, infinite progress along a route is desirable. In other words, an agent should always reach the next station in finite time.

Definition 14 (Partial Path)**.** A partial path is a finite section of an infinite path.

Definition 15 (Path Segment)**.** A path segment is a finite path with the last node being a safe spots and all other nodes not colliding with any safe spot.

An infinite path of an agent is constructed by repeatedly extending a partial path with new segments.

Definition 16 (Progress Function)**.** The progress function $Progress_i(L)$ indicates how much closer an agent a_i gets to its infinite target on its route R when moving along a path segment L.

The condition for infinite progress is:

$$\sum_{j=1}^{\infty} Progress_i(L_j) = \infty$$

To guarantee infinite progress, we need to ensure that every partial path can eventually be extended with a path segment L with $Progress(L) \geq \varepsilon > 0$.

Definition 17 (Infinite Progress)**.** $InfProgress_i(v_{ik})$ denotes that an agent a_i in safe spot v_{ik} can always find a path back to v_{ik}, visiting all stations of its route R and avoiding any other safe spot $v_{jl} \in (K \setminus \{v_{ik}\})$.

$$InfProgress_i(v_{ik}) \coloneqq PathExists(v_{ik}, v_{ik}, K \setminus \{v_{ik}\})$$

The path segment L_{loop}, which we have to find to ensure $InfProgress_i(v_{ik})$, must not necessarily visit the stations in a particular order. We can always construct a segment that visits the stations in any given order, by repeating the loop multiple times.

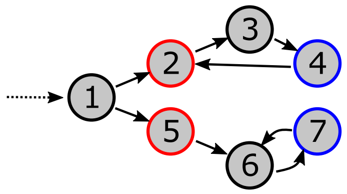

Fig. 2. Movement graph for a single agent with stations $S_1 = \{2, 5\}$ and $S_2 = \{4, 7\}$

Theorem 18. *An agent staying at some safe spot always gets a chance to extend its partial path in finite time.*

Proof. If agents cannot collide and the movement graph is finite, the number of agents must be finite. Any agent in a safe spot takes only finite time for a movement attempt: If there is a path segment guaranteeing progress, moving will only take a bounded time. Otherwise, the agent cannot move, which takes zero time. The total time used for all agents' movement attempts is therefore finite. □

Theorem 19. *An agent a_i moving along route R and staying at safe spot v_{ik} makes infinite progress in infinite time if $InfProgress_i(v_{ik})$ is true.*

Proof. Theorem 18 guarantees that agent a_i, staying at safe spot v_{ik}, is able to attempt extending its partial path in finite time. Definition 17 ensures the existence of a path segment L leading back to v_{ik} and passing through every station $S_i \in R$. Because L passes through all stations $S_i \in R$, the agent always reaches its next station and therefore makes progress. □

Example 20. For the graph shown in Figure 2, initially any node can be selected as a safe spot. By selecting Node 3, we can guarantee $InfProgress(3)$ for the cyclic route $R = \{S_1, S_2, S_1, S_2, \ldots\}$ as $L = \{3, 4, 2, 3\}$ visits every station of route R. When selecting Node 6 instead, we cannot guarantee $InfProgress(6)$, as no path segment exists that fulfills the requirement.

E. Updating Path Segments

Paths consisting of path segments reaching from safe spot to safe spot can be suboptimal in terms of required travel time. An agent might have to slow down at every safe spot and subsequently accelerate again. A solution to this problem is to update path segments before reaching the next safe spot. Like this, agents might never need to actually stop at a safe spot.

Example 21. Assume that the graph in Figure 3 only contains Safe Spot 1. For an agent moving along a cyclic route $R = \{S_1, S_2, S_1, S_2, \ldots\}$, the only valid path segment must be $L = \{1, 2, 3\}$. Infinitely moving over L will lead to infinite progress. To visit $2n$ stations along route R, we need about $3n$ state transitions. If instead, always before the agent moves to Node 1, we update the path segment to a new path segment $L_{new} = \{3, 2, 3, 1\}$, we construct an infinite path $\{1, 2, 3, 2, 3, 2, 3, 2, \ldots\}$. Visiting $2n$ stations along route R now only needs about $2n$ state transitions.

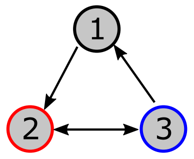

Fig. 3. A movement graph with stations $S_1 = \{2\}, S_2 = \{3\}$

IV. Implementation

Our technique can be used to route different kinds of traffic. We implemented simulations for a road and a naval scenarios. The difference between the scenarios is that in the road scenario, the movement graph is sparser, due to the given roads lanes, while the ships in the naval scenario have much more choices due to a denser movement graph. Due to space constraints, and since the naval scenario is computationally more challenging because of the larger number of choices in the denser graph, we only discuss the naval scenario here.

In our implementation, all agents move simultaneously, if possible. We achieve this by searching paths for all agents one after the other, while excluding nodes occupied by agents whose paths have been computed earlier.

The naval traffic sample application simulates ships moving on cyclic routes between multiple harbors. Waters consist of a regular grid in which each cell is either land or water. The maximum grid size we tested was 512 by 512 cells. A ship's state is composed of:

- Position: Using the same granularity as the grid
- Rotation: 32 different angles
- Velocity: 7 steps including standstill

A ship's state transition is governed by the following rules:

- The ship may accelerate, keep its velocity or decelerate.
- The ship can either turn left, turn right or keep the direction. A complete 360° turn requires 32 state transitions.
- The ship must move to another cell.
- The ship must not collide with land cells or other ships.

The maximum number of states for a scenario with a 512 by 512 grid is $512 * 512 * 32 * 7 = 58720256$, if the whole map consists of water only.

For pathfinding, we use a hierarchical A* algorithm [8] to find path segments. Hierarchical A* uses the result of an A* search in fewer dimensions, in our case only the distance, as a heuristic for the complete A* search. This heuristic is more precise than a simple Euclidean distance heuristic as it takes the topology of the waters into account.

By multiplying the distance value by a heuristic factor, we overestimate the number of steps needed to reach the next station. This is a common technique to trade solution quality for reduced computation time with A*. We experimentally determined that a heuristic factor of 1.7 does not increase the length of the computed path, but leads to a speed-up factor of 158. With that, a path search for one agent took 103 ms on our notebook computer, compared to 16333 ms with the Euclidean distance measure. Since completing a computed path usually takes at least several seconds, our system can thus support many agents in real-time. For road networks, the state space is much smaller and thus even more agents can be handled with our method.

V. Conclusion

Our method enables deadlock-free traffic routing with progress guarantees for many agents in densely populated traffic networks. Our abstract framework can be adapted to different scenarios, including simple cases such as railroads in which the network is already given as a graph and more challenging environments such as ship traffic. With a good choice of the shortest path search algorithm, our traffic scheduler can run in real-time even though collisions during the path search increase with larger numbers of traffic participants.

Today, rail networks are already operated in a tightly regulated fashion similar to the system which we propose. In contrast, car and ship traffic is relatively anarchic. In such systems it is a challenge to efficiently route dense traffic.[7] The advent of self-driving, connected cars and ship localization and tracking systems such as the *automatic identification system (AIS)* makes guided routing a possibility. To cope with the ever increasing traffic volume, this will also become a necessity in congested areas.

The safe spots introduced in this paper allow partitioning traffic networks into independent parts. An agent can be routed to a *border safe spot* in one network and then continue its path in the adjacent network, with the routes in both networks being independent of each other. This also enables decentralized traffic management, for instance by different traffic authorities. In the real world, this can be used as a moderate solution which does not force totally controlled traffic. Our method can be applied to increase the safety in the most frequently used streets and rivers and still allow anarchic traffic in less critical areas.

References

[1] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," in *Numerische Mathematik, 1: 269-271*, 1959.

[2] R. S. Max Barer, Guni Sharon and A. Felner, "Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem," in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[3] T. S. Standley, "Finding Optimal Solutions to Cooperative Pathfinding Problems," in *AAAI*, vol. 1. Atlanta, GA, 2010, pp. 28–29.

[4] G. Wagner and H. Choset, "M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 3260–3267.

[5] P. Surynek, "Towards Optimal Cooperative Path Planning in Hard Setups Through Satisfiability Solving," in *Pacific Rim International Conference on Artificial Intelligence*. Springer, 2012, pp. 564–576.

[6] J. Yu and S. M. LaValle, "Planning Optimal Paths for Multiple Robots on Graphs," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3612–3617.

[7] H. C. L. Teck-Hou Teng and A. Kumar, ""Coordinating Vessel Traffic to Improve Safety and Efficiency"," in *16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, May 2017.

[8] R. Z. Robert C. Holte, M.B. Perez and A. MaxDonald, "Hierarchical A*: Searching Abstraction Hierarchies Efficiently," in *National Conference on Artificial Intelligence, AAAI-96*, 1996.

[7]Ship traffic features various crowded hot spots which can be found in the map at https://www.marinetraffic.com/.