Ship Patrol: Multiagent Patrol in Complex Environmental Conditions

Noa Agmon[1], Daniel Urieli[2], and Peter Stone[2]

Abstract T
he problem of multiagent patrol has gained considerable attention during the past decade, with the immediate applicability of the problem being one of its main sources of interest. In this work we concentrate on frequency-based patrol, in which the agents' goal is to optimize a frequency criterion, namely, minimizing the time between visits to a set of interest points. We consider multiagent patrol in environments with complex environmental conditions that affect the cost of traveling from one point to another. For example, in marine environments, the travel time of ships depends on parameters such as wind, water currents, and waves. We demonstrate that in such environments there is a need to consider a new multiagent patrol strategy which divides the given area into parts in which more than one agent is active, for improving frequency. We show that in general graphs this problem is intractable, therefore we focus on simplified (yet realistic) cyclic graphs with possible inner edges. Although the problem remains generally intractable in such graphs, we provide a heuristic algorithm that is shown to significantly improve point-visit frequency compared to other patrol strategies. For evaluation of our work we used a custom developed ship simulator that realistically models ship movement constraints such as engine force and drag and reaction of the ship to environmental changes.

[1] Department of Computer Science, Bar Ilan University, Israel
agmon@cs.biu.ac.il
[2] Department of Computer Science, The University of Texas at Austin, USA
{urieli, pstone}@cs.utexas.edu

1 Introduction

The problem of multiagent patrol has gained considerable attention during the past decade [6, 10, 3, 8, 4, 2, 1], with the immediate applicability of the problem being one of its main sources of interest. The problem is formally described as repeatedly visiting some interest points in order to monitor them. The points may either be in a discrete environment, a continuous 1-dimensional environment (along a line), or a continuous 2-dimensional environment (inside an area).[1] The problem is usually divided according to the perspective of the agents. In *multiagent frequency-based patrol*, the agents' goal is to optimize some point-visit criterion, for example minimizing the maximal time between visits to a point (e.g. [6, 8]). In *multiagent adversarial patrol* the agents' goal is to maximize their chances of detecting an adversary that tries to penetrate through their patrol path undetected (e.g. [4, 1]).

In this work we concentrate on the continuous 2-dimensional frequency-based multiagent patrol problem, with discrete points of interest, in complex environmental conditions. In this problem, we are given a graph $G = (V, E)$, and we need to define patrol paths for a team of k agents that will minimize the maximal time some vertex of the graph is left unvisited. The complexity of the environment is expressed via the cost of travel between each pair of vertices of the graph.

Consider the problem of ship patrol, i.e., patrol by agents (ships) in marine environments. When designing algorithms for ships in such environments, it is critical to consider the impact of the environment and the specifications of the ship on the behavior of the ship that might also change over time. In our case, we incorporate the shape and engine power of the ship, and environment conditions such as water currents and winds in modeling the environment as graphs, namely in its affect on the cost of travel between vertices of the graph.

Current strategies for multiagent patrol offer, roughly, two alternatives for agents' patrol paths. The first strategy, denoted herein as SingleCycle, is to create one simple cyclic path that travels through the entire area (graph), and to let all agents patrol along this cyclic path while maintaining uniform distance between them [8, 6]. The second strategy, denoted herein by UniPartition, is to partition the area (graph) into k distinct subareas, where each agent patrols inside one area.

We suggest a third, general, strategy, denoted by MultiPartition, in which the graph is divided into m subgraphs, $m \leq k$, such that a subteam of agents jointly patrols in each subgraph. We define the problem of finding k (possibly overlapping) paths for the agents such that the maximal time between any two visits at a vertex is minimized, and show that the problem is $\mathcal{NP}$-Hard. The SingleCycle and UniPartition strategies, as special cases of MultiPartition, are also intractable in general graphs.

[1] Of course higher dimensions are also possible.

An additional version of the problem, in which the graph is to be divided into m *disjoint cycles*, where the k agents are divided among the cycles, is also intractable in general graphs. We therefore investigate the problem on a special family of graphs, which are cyclic graphs with non intersecting shortcuts (diagonals), called *outerplanar graphs* [5]. This simplified, yet realistic, family of graphs have some characteristics that can be of help when looking for optimal solutions to the multiagent patrol problem. For example, an optimal SingleCycle strategy is unique and can be found in linear time. Unfortunately, the time complexity of the general problem of finding an optimal MultiPartition strategy even in such graphs appears to be intractable as well. We therefore suggest a heuristic algorithm HeuristicDivide for finding a partition of the graph into disjoint cycles in the outerplanar marine environment, and a partition of the k agents among those cycles.

For evaluation of our work we used a custom developed ship simulator, UTSeaSim, that was designed to realistically model ship movement constraints in marine environments. UTSeaSim simulates the specifications of the ship, namely, the weight, engine power, and shape of the ship; and how it is influenced by the environmental conditions, including water, currents and winds. We first show that in a simple scenario in which the optimal MultiPartition strategy is easily computable, it outperforms the other two strategies (SingleCycle and UniPartition). We then show that in a more complex environment, our heuristic algorithm HeuristicDivide, following the MultiPartition strategy, performs significantly better than the tractable SingleCycle strategy.

The book chapter is organized as follows. Section 2 describes previous work in the research area of multiagent patrol. In Section 3 we describe the motivation behind the new definition of the MultiPartition strategy, originated in the implementation of the ship-patrol problem in a realistic marine environment simulator. We then provide a new formal definition of the multiagent patrol problem along with its complexity in Section 4, and discuss the problem in outerplanar graphs. In Section 6 we describe the ship simulator and the empirical evaluation of the algorithms we discuss in the work. Last, we conclude along with directions for future work in Section 7.

2 Related Work

The problem of multiagent patrol can be roughly divided into two problems: multiagent frequency-based patrol (e.g. [10, 6, 8]), and multiagent patrol in adversarial environments (e.g. [1, 4]). The problems differ in the objective function that should be optimized, namely optimizing frequency-based criteria or optimizing probability of detecting events controlled by an adversary (respectively). In this work we focus on the problem of frequency-based patrol, in which we aim at minimizing the time between two visits at a point.

Mechado *et al.* [10] were the first to define the problem of multiagent patrol in graph environments, and introduced the notion of *idleness*, meaning the time between two visits in a vertex of the graph. They consider environments with uniform length edges, and perform an empirical evaluation of various architectures for multiagent patrol in different graphs. Generally, they distinguish between reactive and cognitive agents, where the former are locally-driven agents using minimal coordination (if any), and the latter might try to use global state while deciding their next move. They did not theoretically define nor evaluate the multiagent patrol problem on graphs, nor did they consider complex environments.

The first *theoretical* analysis of the problem of multiagent patrol was given by Chevaleyre [6]. Chevaleyre refers mainly to the *worst idleness* criterion, which is the largest amount of time that some vertex remained unvisited throughout the execution of the patrol algorithm. He discusses two possible strategies: a *Cyclic* strategy, in which one cyclic path travels through the entire graph, and all agents follow this path (denoted by SingleCycle) and a *Partition-based* strategy, in which the graph is partitioned into k distinct subgraphs (k being the number of agents), where each agent visits one subgraph in a cyclic tour (denoted by UniPartition). He analyzes the idleness criterion in each of these strategies, using an approximation algorithm to the Traveling Salesman Problem (TSP) under the assumption that the triangle inequality holds. In our work we redefine the multiagent patrol problem in a more general form, in which the graph is possibly partitioned into disjoint subgraphs, however agents can share a subgraph (denoted by MultiPartition). In addition, we do not assume that the triangle inequality holds (as in many realistic scenarios this assumption is not true). This general definition includes also the two strategies proposed by Chevaleyre as subcases, i.e., $\mathsf{SingleCycle}, \mathsf{UniPartition} \subseteq \mathsf{MultiPartition}$.

Ahmadi and Stone [2] investigated the multiagent patrol problem in prioritized environments, i.e., where different areas require different attention from the agents. They suggest a new, learning-based method for determining the optimal patrol path for each robot, which is adapted to the different constraints of the environment. In this work we consider nodes with uniform priority, i.e., all nodes should have minimal possible idleness.

Multi-robot patrol in areas was considered by Elmaliach *et al.* [8], which offered an optimal patrol algorithm using a cyclic strategy, i.e., one cyclic path with minimal cost passes through the entire area, and all robots coordinatedly travel along this path. Their solution assumes that the area and the size of the robots meet several constraints, allowing them to find an optimal solution (minimal cost cyclic path) in polynomial time.

Elmaliach *et al.* [9] considered the problem of frequency-based multi-robot patrol along an open fence, where they evaluated their patrol algorithm according to different frequency criteria. They offer a model for determining the patrol path of the robots in this asymmetric environment, which takes into account the motion model of the robots (acceleration and velocity changes,

and error in motion). This model results in a realistic cost of travel along the fence. In our work we consider the case of uncertain cost of travel that depends on the environment, and is updated during the patrol execution (rather than fixed in advance given the physical constraints of the robots). Moreover, both the general graph model and the restricted outerplanar graph model pose a considerable challenge compared to the linear environment of a fence, due to the number of new possibilities of patrol paths for the robots.

Recent work by Marier *et al.* [11] describe a solution to multiagent patrol on graphs with non-uniform weights on the vertices of the graph, corresponding to the importance of the node. They offer two algorithms for patrolling. The first is reactive (based on consequences calculated for a very limited horizon) and the second is based on an online heuristic algorithm for solving POMDPs. They describe the problem as information gain (rather than idleness), and examine the performance of their heuristic algorithms with respect to the known (or unknown) duration of the patrol. They do not consider uncertainty in travel time, nor do they refer to the graph theoretic problem.

3 Motivation - Ship Patrol and Marine Environment

As surveyed in Section 2, the problem of multiagent patrol has become a canonical problem in multiagent (and specifically multi-robot) systems in the past several years. In this work, we investigate this problem in a realistic ship simulator that we have designed in our lab and that introduces important new travel-time constraints to the problem (a technical description of the simulator is given in Section 6). The general problem defined in graph environments requires a team of k agents to repeatedly visit all N nodes of the given graph while minimizing the longest time a node has remained unvisited by some robot. We first look into the simplest scenario found in the literature, namely patrol in circular environments. In such environments, the patrol path is linear and the algorithm that optimizes point-visit frequency was shown to be an algorithm that requires all robots to travel in a coordinated manner and maintain uniform (time) distance between each neighboring pair of robots along the path (e.g. [6, 8]).

In marine environments, as in other complex realistic environments, two factors have the most influence on the travel time of a ship: specifications of the ship, and the conditions of the environment, namely the sea. When designing an algorithm for such environments, we therefore have to take into account these factors, that also might change during mission execution (the patrol).

After implementing a simple scenario, in which three ships patrol along a cyclic path in order to optimize point-visit frequency over a set of 10 points (see Figure 1), we discovered several interesting phenomena. First and foremost, we saw that in environments in which the cyclic path is not along a

perimeter of some closed structure (for example airports, prisons and military bases), it is necessary to consider paths that create shortcuts between point of interest, and allow traveling from one point to another that are not necessarily along the cyclic path. Moreover, even in such environments (especially military bases, factories and airports), there usually are roads going through the closed area, creating shortcuts in transitions between points along the perimeter. Second, when we applied different water current and wave conditions, the cost of traveling (corresponding to travel time) along some edges of the graph became very high, encouraging the use of the shortcuts for traveling between points of interest. We examined solutions that exist in the literature for defining optimal patrol paths for a team of robots, and found only the SingleCycle and UniPartition solutions, which consider the entire cyclic path, or divide the patrol paths into k areas, each under the responsibility of a single agent (respectively).

In the example illustrated in Figure 1, we examined two scenarios. In the first we had no currents or winds (a "clean" environment), and in the second we introduced winds and currents, specifically currents between points p_2 and p_3, and between points p_6 and p_7. The travel time between p_2 and p_3 and between p_6 and p_7 (in both directions) is, therefore, very high. The *worst idleness* results we describe here are calculated by averaging the idleness of each point along a 10 minute execution of the patrol in the simulator, and choosing the point with highest average idleness value. In the first (clean) environment, when the three ships executed the SingleCycle strategy, we got an average worst idleness of 651 seconds. When dividing the set of 10 points among the three ships, where one ship patrols along points p_3, p_4, p_5, p_6 in a circular path, and the other two ships divide the remaining points between them (the UniPartition strategy) we get worst idleness of 786 seconds. However, when looking closely at this example, it can be clearly seen that there exists another possibility: letting one ship patrol along p_3, p_4, p_5, p_6, and having the other two share the cycle $p_1, p_2, p_7, p_8, p_9, p_{10}$ and patrol, coordinatedly, with uniform time distance between them (the MultiPartition strategy). By executing this algorithm, we got worst idleness of 614 seconds, a major improvement compared to the previous two strategies. This improvement becomes more substantial when we examine the situation with currents. Now, the SingleCycle strategy yields worst idleness of 795 seconds, the UniPartition yields worst idleness of 792 seconds, and the MultiPartition strategy yields worst idleness of 613 seconds (note that in the last two cases there is no significant change from the clean environment, as the ships did not travel through the stormy weather, i.e., where the strong currents are).

This example, along with other similar phenomena we have viewed in our simulator, motivated us to redefine the problem of multiagent patrol in a more general form, denoted as MultiPartition, and discuss possible solutions to the problem in circular environments, but with additional shortcuts between the points of interest.

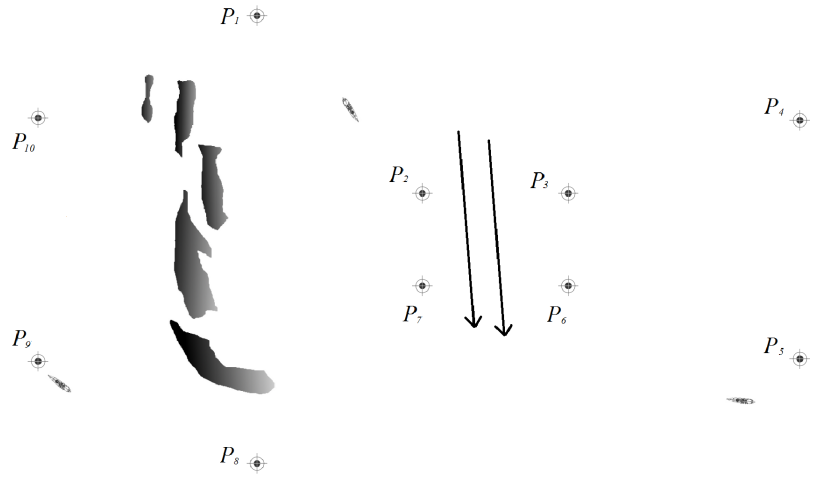

Fig. 1 An example of a scenario handled by the simulator. The circles represent the points of interest (nodes of the graph), and the drop shapes are the ships. The large grey shapes are obstacles, and the drawn arrows indicate the direction of the water current.

4 Problem definition and complexity

In this section, we define the general problem of multiagent frequency based patrol on general graphs. We describe the decision version of the problem, where the input is the graph $G = (V, E)$ ($|V| = N$), an integer $k < N$ that corresponds to the number of agents, and an integer f which is the maximal worst idleness, i.e., the maximal requested idleness from all vertices of the graph (similar to the definition in [6]). Formally, if f_i denotes the idleness of a vertex v_i, then the worst idleness of the graph G, $\mathrm{wi}(G)$, guaranteed by an Algorithm $\mathcal{A}$ is defined as $\mathrm{wi}(G) = \max_{1 \leq i \leq N}\{f_i\}$.

Note that real world constraints dictate modeling the world with directed graphs, i.e., the travel time from a vertex v to a vertex u is not necessarily the same as that from u to v. However, we assume that the graph is *symmetric*, i.e., if an edge exists from v to u, then an edge exists also from u to v (not necessarily of the same cost). We therefore describe the general problem on undirected graphs. Once a cycle is defined, the algorithms will decide whether to go clockwise or counterclockwise along the cycle, depending on the direction that has lower cost.

4.1 Multiagent patrol in general graphs

Definition: Multiagent Graph Patrol (MGP)
Given a graph $G = (V, E, C)$ where $|V| = N$, and $\forall (v_i, v_j) \in E, c_{ij} \in C$ is the associated cost of the edge, an integer $k < N$ denoting the number of agents, and a desired maximal worst idleness target f, is there a division of V into $m \leq k$ cyclic paths $V_1, V_2, \ldots, V_m$ (not necessarily simple), each assigned with k_i agents such that all k_i agents visit all vertices in V_i and $\sum_{i=1}^{m} k_m = k$, such that the worst idleness wi(G) is at most f?

In the following theorem we show that the MGP problem is $\mathcal{NP}$ complete for general k's.

Theorem 1. *The* MGP *problem is* $\mathcal{NP}$ *complete.*

Proof. First, the MGP problem is in $\mathcal{NP}$, since given a solution, i.e., a division of the graph into m paths, it is possible to verify whether wi(G) is indeed f using a variation of the Algorithm AssignKAgents. MGP is $\mathcal{NP}$-Hard for general k by a simple Turing reduction from the decision version of the graphical traveling salesman problem ($GTSP$). Specifically, we can find whether there exists a minimal tour of size at most f that travels through all nodes in the graph at least once by solving the MGP problem given f as input, and $k = 1$.

In our work, we would like to consider a special case of the MGP problem, in which each path $V_1, \ldots, V_m$ is a *simple* cycle, i.e., it is a closed path with no repeated vertices. Moreover, we restrict our attention to sets of distinct paths that do not share any vertices, i.e, $V = V_1 \bigoplus V_2 \bigoplus \ldots \bigoplus V_m$ (distinct simple cycles). This problem handles restrictions that are more suitable for realistic robotic environments, in which two requirements are met:

1. Two robots will not meet during the execution of the algorithm, thus will not interfere with one another during the patrol.
2. Robots will not need to interact outside of their subteam, i.e., the patrol algorithm requires only local coordination (unless the environment changes the optimality of the current patrol algorithm). Moreover, if different human operators observe each subteam, it does not require continuous coordination among the human operators.

The formal definition of the problem is as follows.

Definition: Multiagent Cyclic Graph Patrol (MCGP)
Given a graph $G = (V, E, C)$ where $|V| = N$, and $\forall (v_i, v_j) \in E, c_{ij} \in C$ is the associated cost of the edge, an integer $k < N$ denoting the number of agents and a desired maximal worst idleness target f, is there a division of V into $m \leq k$ *distinct simple* cycles $V = V_1 \bigoplus V_2 \bigoplus \ldots \bigoplus V_m$, each cycle V_i assigned with k_i agents coordinatedly traveling along V_i and $\sum_{i=1}^{m} k_m = k$, such that the worst idleness wi(G) is at most f?

The MCGP is a special case of the MGP, in which the cyclic paths are required to be disjoint, and each cycle is simple (with no repeated vertices). The $\mathcal{NP}$-Hardness proof resembles the proof for the MGP problem, thus we conclude the following.

Corollary 1. *The* MCGP *problem on general graphs is $\mathcal{NP}$-Hard.*

We can define the worst idleness in this problem as follows. If k' agents visit a cyclic path, denoted by V^C, where $V^C = \{v_{i_1}, v_{i_2}, \ldots, v_{i_l}\}$, $v_{i_j} \in V(G)$, $(v_{i_j}, v_{i_{j+1 mod l}}) \in E(G)$, and denote the total weight of edges in the cycle by $w(V^C) = \sum_{h=1}^{l} c_{i_j i_{(j+1 mod l)}}$, then $\forall v_{i_j} \in V^C$, $f_{i_j} = \frac{w(V^C)}{k'}$. Therefore if G is divided into m distinct cycles, where each cycle V_i^C is visited by k_i agents, then $\text{wi}(G) = \max_{1 \le i \le m}\{\frac{w(V_i^C)}{k_i}\}$.

Algorithm *AssignKAgents* (described below) is given as input m cyclic paths, an integer k corresponding to the number of agents, and a maximal idleness f, and has to answer the question of whether k agents are sufficient to guarantee a maximal idleness of f on the given graphs. It returns the assignment of number of agents per graph ($K = \{k_1, \ldots, k_m\}$ such that $\sum_{i=1}^{m} k_i = k$ and k_i agents are necessary to visit G_i in order to guarantee minimal idleness f) and the maximal idleness guaranteed by this assignment (f_{loc}). Denote the edges along the cyclic path G_i in clockwise direction by G_i^{cw} and in the counterclockwise direction by G_i^{ccw}. The algorithm will work for either symmetric directed graphs (in which it will refer to the direction with minimal cost — either going clockwise or counterclockwise) or undirected graphs (in which $w(G_i^{cw}) = w(G_i^{ccw})$ where $w()$ is the cycle weight, or length, function).

Algorithm 1 $< K, f_{loc} >$ = Algorithm AssignKAgents$(\{G_1, \ldots, G_m\}, k, f)$

$C \leftarrow 0$, $K \leftarrow \emptyset$
for $i \leftarrow 1, \ldots, m$ **do**
  $w_i \leftarrow min\{w(G_i^{cw}), w(G_i^{ccw})\}$
  $c_i \leftarrow argmin_{G_i^{cw}, G_i^{ccw}}\{w(G_i^{cw}), w(G_i^{ccw})\}$
  $k_i \leftarrow \lceil w_i / f \rceil$
  if $k_i > k$ **then**
    Return $\emptyset$
  $K \leftarrow K \bigcup k_i$, $C \leftarrow C \bigcup c_i$
  $k \leftarrow k - k_i$
$f_{loc} \leftarrow \max_{1 \le i \le m}\{w(C[i])/K[i]\}$
Return K, f_{loc}

4.2 The multiagent patrol problem in outerplanar graphs

Motivated by the problem of multi-robot *perimeter patrol* (e.g. [1]), we examine the MCGP problem in circular environments. However, we add more realistic considerations to the environment, namely adding possible *shortcuts* between vertices that pass inside the circle. To avoid possible interference by agents that travel along the edges, we require the inner edges not to intersect one another. The resulting graph is planar, and moreover, it is a *biconnected outerplanar* graph [5], i.e., it is a planar graph that is cyclic, and there are no nodes that are inside the cycle (all nodes in the graph are on the same outer face).

An example for such a graph is shown in Figure 2. In this example, if an edge existed between v_4 and v_{11}, then the graph would not be planar (as the edge (v_5, v_{11}) crosses the edges (v_6, v_{12}) and (v_8, v_{12})). Also, if an edge existed between v_{13} and v_{11}, then the graph would remain planar, but would not be outerplanar (as v_{12} is not adjacent to the outer surface anymore).

In the family of outerplanar graphs, several hard problems become very easy to solve. For example finding a Hamiltonian cycle is done in linear time, as the only possible simple cycle that visits all nodes in the graph is the external cycle. Therefore also finding the optimal SingleCycle strategy is done in linear time, as the solution is unique. Another interesting characteristic of outerplanar graphs is that every subgraph of an outerplanar graph is outerplanar, thus a biconnected component of a subgraph of an outerplanar graph also has a unique Hamiltonian cycle ([5]).

We draw a connection between disjoint cycles and biconnected components in Lemma 1. Generally, a biconnected component in an outerplanar graph has a unique Hamiltonian cycle, which is the outer-cycle that visits all vertices. We would therefore like to find a way to use this property in order to find disjoint cycles, as defined in the MultiPartition strategy. As a first step, we look at the case of dividing the graph into two disjoint cycles. We show in the lemma that in order to find such disjoint cycles, it is sufficient and required (in the general case) to consider all biconnected components that are created by the removal of two edges from the graph. We later use this property in the heuristic algorithm for solving the MCGP problem in outerplanar graphs.

Consider the outerplanar graph $G = (V, E)$ in Figure 2. We will demonstrate why the removal of only one edge might not result in all disjoint cycles, and how we can achieve either two or three disjoint cycles by removing every pair of edges. First, by removing only one edge $e \in E$ and computing the biconnected components in the remaining graph $G = (V, E \setminus \{e\})$ it would be impossible to get the division of the graph into the two disjoint cycles $V_1^C = (v_9, v_{10}, v_{11})$ and $V_2^C = G \setminus V_1^C$: Removing (v_8, v_9) would result in the biconnected components $\{v_9, v_{10}, v_{11}\}$ and $\{v_1, \ldots, v_8, v_{11}, v_{12}, \ldots, v_{15}\}$, which are not disjoint, thus their Hamiltonian cycles are not disjoint. The

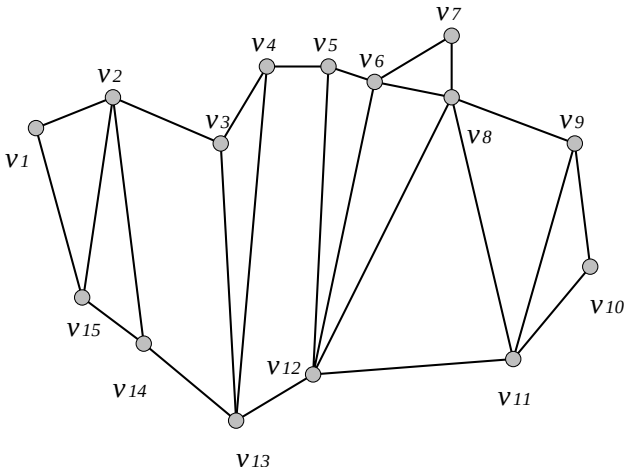

Fig. 2 An outerplanar graph: The vertices of the graph (points of interest for the agents along the patrol) are all adjacent to one external face, and the inner edges (shortcuts) do not intersect.

removal of (v_8, v_{11}) results in one biconnected component (thus cycle) G, and the removal of (v_{11}, v_{12}) results in the cycles $\{v_8, v_9, v_{10}, v_{11}\}$ and $\{v_1, \ldots, v_7, v_8, v_{12}, \ldots, v_{15}\}$, which are again not disjoint. Therefore by removing only one edge we could not get the disjoint cycles V_1^C and V_2^C. However, by removing (v_8, v_9) and (v_8, v_{11}), this division is achieved. Note that the removal of the pair of edges (v_2, v_3) and (v_4, v_5) results in three disjoint biconnected components (thus cycles): $\{v_1, v_2, v_{14}, v_{15}\}$, $\{v_3, v_4, v_{13}\}$ and $\{v_5, v_6, \ldots, v_{12}\}$.

Lemma 1. *Given a biconnected outerplanar graph $G = (V, E)$, each division of G into two disjoint biconnected components can be achieved by removing one pair of edges and computing the biconnected components of the remaining graph. If removing one pair of edges, the number of remaining disjoint biconnected components (excluding disconnected vertices) will not exceed* 3.

Proof. We first show that two disjoint biconnected components $V_1^C = \{v_1, \ldots, v_l\}$ and $V_2^C = \{u_1, \ldots, u_h\}$ in an outerplanar graph G can be connected by either two or three edges. First, assume that V_1^C and V_2^C are connected by only one edge (v_i, u_j), $v_i \in V_1^C$ and $u_j \in V_2^C$. Therefore both v_i and u_j are articulation vertices (their removal disconnects the graph), contradicting the nonseparability characteristic of the biconnected graph G.

We now show that V_1^C and V_2^C cannot be connected by more than three edges. Without loss of generality, assume that both V_1^C and V_2^C vertices are ordered clockwise in ascending order, and that V_1^C is left of V_2^C. Therefore, since G is outerplanar, two edges connecting V_1^C and V_2^C are necessarily (v_i, u_j) and (v_{i+1}, u_{j+1}). Moreover, there cannot be any other edge (v, u)

connecting V_1^C and V_2^C such that $v \notin \{v_i, v_{i+1}\}$ and $u \notin \{u_j, u_{j+1}\}$, otherwise some vertex $v'_i \in V_1^C$ and/or $u'_j \in V_2^C$ are not adjacent to the outer face, contradicting the outerplanar definition of G. Therefore the only possible edges connecting the two disjoint components are between vertices v_i, v_{i+1} and u_j, u_{j+1}. Since an outerplanar graph cannot have a subgraph that is a clique of size 4 [5], and since necessarily $(v_i, v_{i+1}) \in E$, $(u_j, u_{j+1}) \in E$ and we've shown that $(v_i, u_j), (v_{i+1}, u_{j+1}) \in E$, then there could exist only one more edge connecting the two components: either (v_i, u_{j+1}) or (v_{i+1}, u_j), but not both.

Since two disjoint biconnected components can be connected by at most three edges, by removing every possible pair of edges from the graph, at some point we will necessarily remove two of the connecting edges of V_1^C and V_2^C, resulting in two disjoint biconnected components. Moreover, the removal of one edge can result in dividing the graph into two disjoint components only if these components are connected by only two edges. Therefore, removing a pair of edges can result in up to three biconnected components (cycles) and no more than that.

This lemma results in the fact that finding two disjoint cycles (and possibly 3) in a graph can be done efficiently in time complexity of at most $\binom{|E|}{2}$. Since finding the partition of k into two (or three) components is done efficiently as well, the MCGP problem can be solved optimally in polynomial time if m is restricted to be 2.

Corollary 2. *In an outerplanar graph $G = (V, E)$, finding a division of the graph into up to two disjoint simple cycles V_1^C and V_2^C such that $V = V_1^C \bigoplus V_2^C$ and $wf(G)$ (for any value of k) is minimized can be done in polynomial time, using Algorithm* DivideTo2Cycles.

Algorithm DivideTo2Cycles receives as input the graph $G = (V, E)$ and the maximal frequency criterion f that should be met, and returns the best division of the graphs into two components such that the division results in maximal idleness of at most f. If no such division exists, it returns the empty set. Note that in order to get all possible divisions of G into two disjoint cyclic paths, the algorithm should be given the value $f = w(G)/k$. The algorithm removes all possible pairs of edges from the original graph, and computes the biconnected components of the remaining graph. For those biconnected components, it checks whether there is an assignment of k into those biconnected components such that the maximal idleness of the graph is at most f, using Algorithm AssignKAgents. By Corollary 2, the algorithm examines all possibilities of dividing the graph into two cycles (which has time complexity of $\mathcal{O}(|E|^2)$). Since checking all possibilities of partitioning the number k into at most 3 components is polynomial in k, and determining the idleness is linear in $|E|$, then the total time complexity of Algorithm DivideTo2Cycles is $\mathcal{O}(|E|^3)$.

As shown by de Mier and Noy [7], the number of cycles in a maximal outerplanar graph is exponential in the number of vertices of the graph, thus

Algorithm 2 S = Algorithm DivideTo2Cycles$(G = (V, E), f, k)$

1: $S \leftarrow \emptyset$
2: **for** Every pair of edges $e_i = (v_i, u_i)$ and $e_j = (v_j, u_j)$, $e_i, e_j \in E$ **do**
3: $E' \leftarrow E \setminus \{e_i, e_j\}$
4: $U \leftarrow$ biconnected components of $G' = (V, E')$
5: **if** $< K, f_{loc} >=$ AssignKAgents$(U, k, f)! = \emptyset$ **then**
6: $f_{loc} \leftarrow$ optimal assignment of k agents to U
7: $S \leftarrow S \bigcup \{< U, K, f_{loc} >\}$
8: Return $S[i]$ for which f_{loc} is minimal

if we need to examine all possible sets that generate a direct sum of V it will result in at least an exponential time complexity. We therefore believe (although not proven) that the MCGP problem, also in the simple biconnected outerplanar environment, is intractable, as the number of all possibilities of the division of the graph into up to k subgraphs grows exponentially with k.

We therefore describe a heuristic algorithm, Algorithm HeuristicDivide, for finding a division of the graph into disjoint cycles.

4.3 Heuristic algorithm for multiagent patrol in outerplanar graphs

We describe in this section a heuristic algorithm, Algorithm HeuristicDivide, for finding a set of any number of disjoint cycles in an outerplanar graph (based on Algorithm DivideTo2Cycles), and dividing the k agents between these disjoint cycles in a way that aims to find a low overall maximal idleness.

The algorithm applies algorithm DivideTo2Cycles once, then further applies itself recursively on each element of the set of disjoint biconnected components that yield lowest idleness. In this way it performs a greedy heuristic search and halts once it completes all possible divisions up to k cycles. The algorithm receives as input the graph G, the number of agents k, and $f = \frac{w(G)}{k}$.

Note that once the cycles, the direction of travel along the cycles, and the division of the agents among the cycles are determined, it is only left to distribute the agents along the cycles (number of agents per cycle as determined by the algorithm), and require the agents in each cycle to maintain uniform distance between them and continue traveling coordinatedly along their circular path.

Time complexity of Algorithm HeuristicDivide**:**
The time complexity of HeuristicDivide is defined by two components: The depth of the search and the time to process each level of the search tree. Since at each step we deepen the recursion we lose at least one vertex (as the minimal division to two distinct cycles is to a vertex v and to a cyclic graph $G \setminus \{v\}$), the depth of the tree is at most $k-1$. The time complexity of

Algorithm 3 Algorithm HeuristicDivide$(G = (V, E), f, k)$

1: $ResSet \leftarrow$ DivideTo2Cycles(G, f, k)
2: **if** $ResSets = \emptyset$ **then**
3: Return G
4: $CurChoice = argmin_{<U_i,K_i,f_i>\in ResSet}\{f_i\}$
5: Return $\bigcup_{G_i \in U(CurChoice)}$ HeuristicDivide(G_i, K_i, f_i)

AssignKAgents is linear in the number of edges, and the complexity of finding biconnected components is also linear in outerplanar graphs. Therefore the complexity of examining each pair of edges is $\mathcal{O}(|E|)$. When we go down in the recursion, if E is divided into up to three disjoint biconnected components E_1, E_2 and E_3, then we have complexity of $\mathcal{O}(|E_1|^2 + |E_2|^2 + |E_3|^2)$ where $0 \leq |E_1| \leq |E_2| \leq |E_3| < |E|$ and $|E_1| + |E_2| + |E_3| = |E|$. Therefore, since $\mathcal{O}(|E_1|^2 + |E_2|^2 + |E_3|^2) < \mathcal{O}(|E|^2)$ it leads to a total time complexity of $\mathcal{O}(k|E^2|)$.

5 UTSeaSim

UTSeaSim[2] is an open-source, custom-designed naval surface navigation simulator. It uses realistic $2D$ physical models of marine environments and sea vessels, and runs both in GUI and in non-GUI modes. UTSeaSim is written in python, and has a flexible design that allows to easily extend it and plug-in new functionality. Figure 3 shows a snapshot taken during a GUI-based simulation.

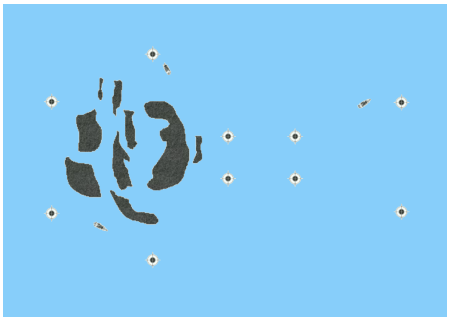

Fig. 3 UTSeaSim's GUI visualization of navigating ships (2d, top-view).

The simulator's core contains three main modules: a *Sea Environment* module, a *Ship* module, and a *Decision-Making* module. The sea environ-

[2] `http://www.cs.utexas.edu/~UTSeaSim`

ment module includes models of winds, water currents, waves, and obstacles. The ship module models all relevant aspects of a ship, including the ship's physical properties, sensing capabilities, and ship actuators. The decision-making module implements an agent that controls a ship autonomously, as well as communicating with other agents to coordinate strategies. At each time step, the agent receives the perceptions sensed by the ship, processes them to update its current world state, and decides on control actions for the ship based on its current world state and its decision-making strategy. The following sections describe the simulation flow and the main simulation modules.

5.1 Simulation Flow

We start with describing the high-level flow of computation, including the inputs and outputs of the simulator. A simulator's run simulates a specific *task*, which is defined in a *task definition file*. The simulator's high-level flow can be described as follows:

- Load the task and task-related data from a task definition file.
- Load environmental model from input configuration files.
- Initialize ships and ship-controlling agents for the loaded task, based on command-line flags.
- Run simulation for n steps. A simulation step advances the world state as a result of agents' actions and exogenous changes in the environment. Simulation can run both in GUI and in non-GUI modes, depending on a command-line flag.
- Write simulation results to file. In general, results could be any simulation data that is of interest to the user and can be gathered during the simulation. The type of results to be written are determined by a command-line flag.

In a typical simulation, autonomous ships are navigating in the sea, executing some task. Each simulation step simulates the world change after one second. During the simulation, data is gathered and is written to output files. Below are the inputs and outputs of a simulation.

5.1.1 Inputs

The inputs to the simulator are environmental conditions and a task definition file:

- **Environmental conditions:** Environmental conditions are defined in text-based files that choose and parametrize an environmental model, including water current directions and speeds and wind speeds, in different

geographic areas. Customizing new environmental models can easily be done through text files.
- **Task definition file:** The task definition file contains a choice to execute one of the supported tasks (an open list that can be easily extended), along with any task-related data, such as patrol points, initial ship positions, and any other task-dependent data.

5.1.2 Outputs

In general, the simulation can output any data that is gathered during the ship navigation simulation. Currently, it can output:

- **Traversal Times Graph Data:** Contains all the patrol nodes, along with a list of the travel times between pairs of points.
- **Point Visit-Frequencies:** This data is gathered during a multi-agent patrol, and maps each patrol node to the average frequency in which it was visited by the patrolling ships.

5.2 Main Simulator Modules

In this section we describe in more detail the core simulation modules.

5.2.1 Sea Environment Module

The sea environment implements different models of winds and currents that affects the ship's motion. The environment conditions model is currently encapsulated inside a Sea class. This class is nothing but a container for Wind, WaterCurrents, Waves and Obstacles classes. Each of the first three classes has only one function: getSpeedVectorInLocation(), which returns, for location (x,y), the speed vector of the wind, water, or the waves respectively. Currently a few simple models are implemented, in which the wind is static and constant, and also the currents are static and constant, but can be different in different geographic areas of the sea. The fourth class, namely Obstacles, is a container of polygon-shaped obstacles.

5.2.2 Ship Module

The ship module models the ships' physical properties, motion modeling and perception capabilities. The Ship class has properties like mass, length, engine force, and proportionality constants that affect the computation of the drag forces and accelerations operating on the ship.

Ship's Perception Capabilities

Our sensing model for the environment is encapsulated inside a ship. The simulator sends its full world state to the ship, and the ship, depending on it's model, extracts from it only the data that it percepts can give it, and send it to the agent that controls the ship. In general, any perception module can be plugged into any ship.

Ship's Motion Model

At every simulation step, we compute the ship's state in the next time step, based on the current world state (the environment), the ship's engine and steering, and the time passed. Currently, we approximate ship movement using the following model:

- For forward motion, we model forward force that operates on the ship by the engine, and a drag, which is quadratic in the ship's speed.
- For turning, there is a rotational torque that is applied by the rudder, and is proportional to the ship's speed, and to the projection of the rudder on the lateral direction. There is also a rotational drag force, that is quadratic in the ship's angular speed.
- Based on the above forces and the ship's mass, we compute the forward and angular accelerations.
- Then, based on the average forward and angular speed in a given time step, computed using the above accelerations, we infer the turn radius, and based on that, compute the ship position at the end of this time-step.

Some approximations we make:

- Although the angular acceleration depends on forward speed, which keeps being changed, we still assume constant forward speed (the avg. speed in this time step) during the computation of angular acceleration.
- Forward acceleration computation does not take into account the effects of turning which might slow it down.
- Forward acceleration depends on the drag, which is changing with speed change, but we approximate the drag based on the initial speed of a time step

A ship's response to the sea conditions is computed inside the ship itself. Each ship has a function getOffsetByEnvConditions() that is responsible for computing the ship's offset resulting from the ship's previous state and the environmental conditions in its (x,y) location. Currently the computation is done based on the ship's orientation, the wind direction, and the currents direction. The environmental effects model is somewhat simplistic: the wind and the current just offsets the ship in their direction, proportionally to their speed, each with a different proportionality constant.

5.2.3 Decision Making Module

The decision making module is encapsulated inside an Agent class. An agent repeatedly processes percepts, updates its belief about the world state and uses its decision making strategy to choose actions to take (usually steering and engine commands to the ship). In order to make decisions, the agent can use a communication module to communicate with other agents. The agent is composed of two main parts: its world model (class AgentWorldModel) and its decision making strategy (class AgentStrategy). In order to implement a new agent, one simply needs to inherit one or both of these interface classes. We will briefly describe them next, along with some other features of the decision making module.

World Model

The world model is responsible for processing the percepts coming in from the ship, and for building a state of the world, possibly in an incremental manner based on the agents beliefs and perceptions. As an example for an agent world model, a world model that is currently implemented is the AgentWorldModelCompleteWorld, which models a fully observable environment. This model assumes that the ship has perfect perceptions and that it receives the complete world state every cycle, so no belief maintenance is needed. In the future, we will add world models with partial observability and sensing of the environment.

Decision Making Strategy

The decision making strategy uses the existing world state that is built and maintained by an AgentWorldModel class, and use it to decide on actions, based on the agent's goals. Two examples for decision making strategies are a cyclic patrol strategy, and a dynamic, coordinated patrol, with the options of ships joining or leaving the patrol.

Communication

An agent can communicate with other agents for planning and executing its task. The interface function getOutgoingMsg() is used by the simulation environment to extract an agent's outgoing messages and deliver them to the specified recipients. Delivering is done by calling the recipient's receiveMsg() function. Besides sending a message to another agent, an agent can send a message to the simulation environment itself, usually to report some statistics.

Obstacle Avoidance

Obstacle avoidance behavior is built into the navigation-related decision making. When a ship starts its navigation towards the next way point, it checks whether the path is blocked by obstacles. If the path is not blocked, the ship navigates directly to the point, using a PID controller (`http://en.wikipedia.org/wiki/PID_controller`) for the steering and engine commands. If the path is blocked, an RRT algorithm (`http://msl.cs.uiuc.edu/rrt/`) computes a bypass and navigates through it.

Rules of the sea

The decision making module has a basic implementation of obeying the rules of the sea. In brief, a ship computes whether there is a potential collision on its navigation path, and in case needed, the ship takes a preventive action. The preventive action is in general: if there are no ships on the right then turn right, otherwise stop.

6 Empirical evaluation

We evaluated HeuristicDivide under realistic marine environment using the UTSeaSim Simulator. In the following section we describe the experimental setting and the empirical results from executing the patrol algorithms described above.

In our experiments, we examined several different environments. Due to space constraints, we describe one interesting environment, in which the graph is outerplanar with a large number of possible division of the graph into disjoint cycles. The marine environment was implemented in UTSeaSim, and illustrated in Figure 4 (distances are shown in meters). In this environment, $N = |V| = 36$, and the number of ships (k) varies from 1 to 30. Although the points of interest are arranged in two straight lines, this environment can become equivalent to many realistic cases by controlling the currents between nodes thus controlling the edge lengths. Moreover, this environment can represent a man-made group of points of interest, for instance a sequence of oil rigs.

We examined four different scenarios, where in each one the sea conditions were different. In the first environment (denoted as *Sea Condition* 0), there were no currents and the cost of traveling between two points correlates to the geographical distance. We then gradually added more currents to the system with three different strengths - *Sea Condition* $1, 2$ *and* 3, for weak, medium and strong, respectively, where their directions was as shown in Figure 4. We ran Algorithm HeuristicDivide initially with the worst idleness of ∞ and let it

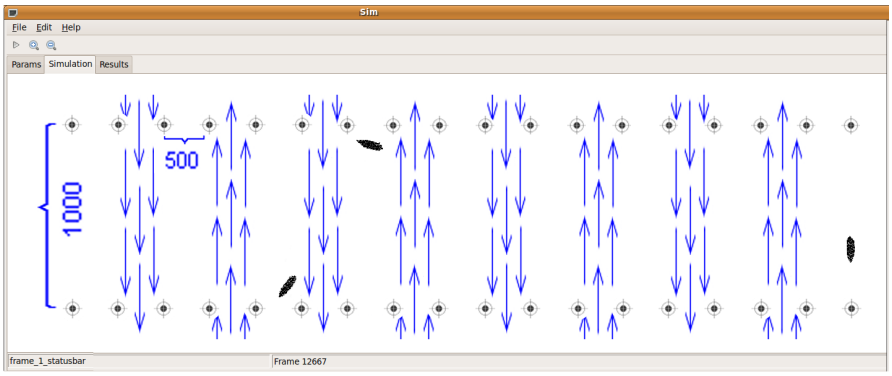

Fig. 4 The evaluation environment.

find the best division it can. We then simulated the patrol-division returned by the algorithm for 20,000 seconds (333 minutes), and reported the worst node-idleness, i.e., the highest average time between consecutive node visits, for any node. Note that in all the results, lower values are *better*.

Figure 5 shows the worst idleness resulting from HeuristicDivide when running on sea conditions 0–3. Note that in Sea Condition 0 (no currents) the algorithm always returned the SingleCycle strategy, which is indeed the best division for the case of no currents. As expected, the worst idleness becomes higher as the sea conditions become rougher.

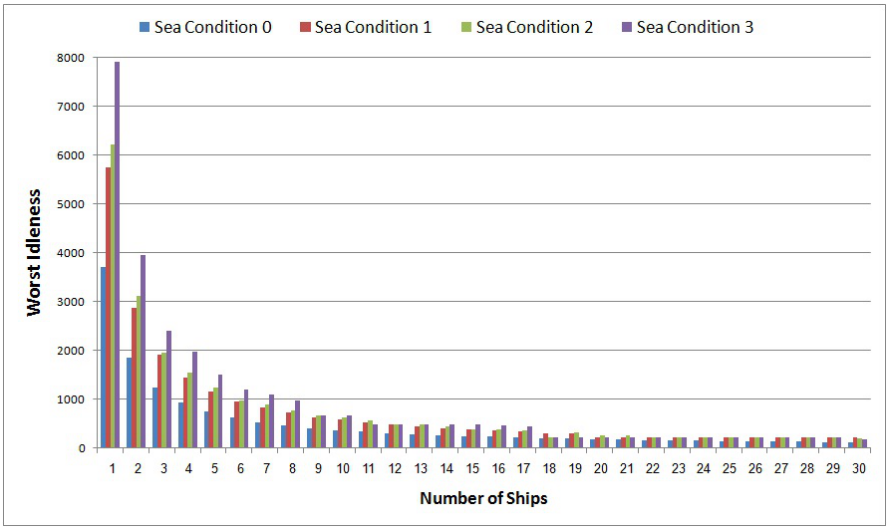

Fig. 5 The worst idleness returned from Algorithm HeuristicDivide in the four different sea conditions examined.

In order to evaluate the performance of our algorithm, we compared its worst idleness with the worst idleness of the following:

1-Loop The result of the SingleCycle strategy, i.e., a patrol algorithm that instructs all ships to patrol around the cycle while maintaining uniform distance between adjacent pairs.

k+1 Incremental change. In this case, we assumed the HeuristicDivide algorithm was computed for k ships, and upon the arrival of a new ship it is added to the cycle with highest worst idleness (for the best improvement in idleness). This is compared to running HeuristicDivide with $k+1$ ships, and the goal is to check whether HeuristicDivide, as a heuristic algorithm, does better than just a straightforward increment.

k-1 Decremental change. In this case, we assumed the HeuristicDivide algorithm was computed for $k+1$ ships, and a ship needs to be removed from the system. We assume that in this case a ship is removed from the cycle with best worst idleness (for minimal increase in the worst idleness). This is compared to running HeuristicDivide with $k-1$ ships, again with the goal of checking whether HeuristicDivide does better than just a straightforward adjustment.

The results are shown in Figures 6, 7 and 8 for Sea Conditions 1, 2 and 3, respectively. Note that since for Sea Condition 0 HeuristicDivide's solution is always the SingleCycle strategy, for every given k HeuristicDivide's solution is identical to the $1-$loop, $k+1$ and $k-1$ solutions.

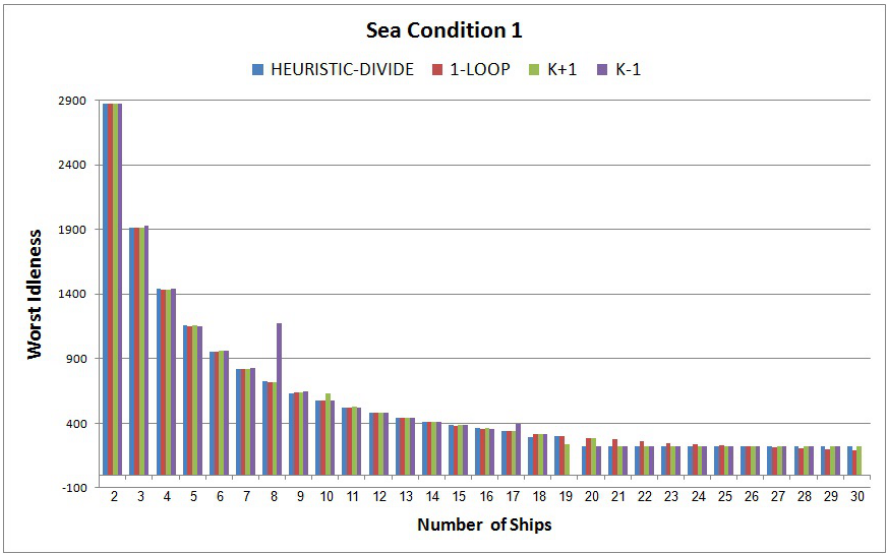

Fig. 6 The results of HeuristicDivide, $1-$loop, $k+1$ and $k-1$ in Sea Condition 1 (weak currents).

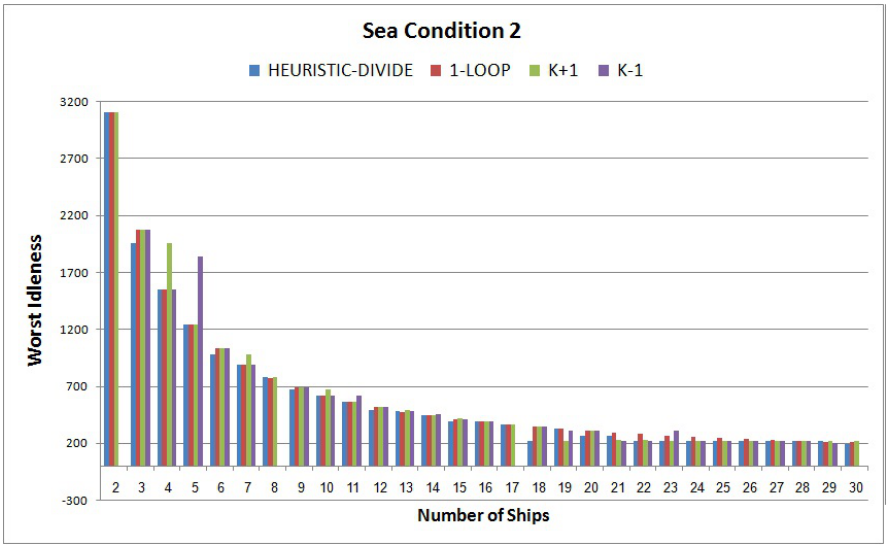

Fig. 7 The results of HeuristicDivide, 1−loop, $k+1$ and $k-1$ in Sea Condition 2 (medium currents).

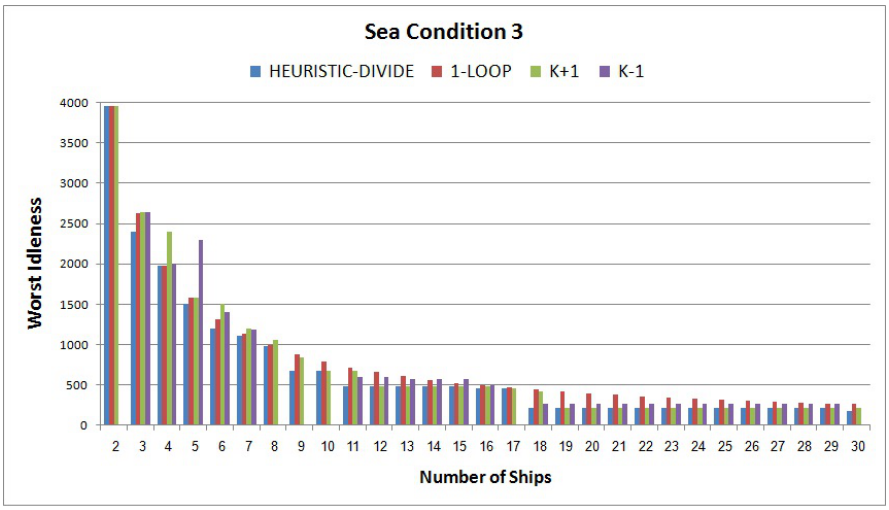

Fig. 8 The results of HeuristicDivide, 1−loop, $k+1$ and $k-1$ in Sea Condition 3 (strong currents).

In Sea Conditions 2 and 3, algorithm HeuristicDivide performs statistically significantly (using paired t-test) better than the 1−loop algorithm and the incremental $k+1$ and decremental $k-1$ cases, with p-values < 0.002 for Sea Condition 3 (in all cases) and p-value < 0.04 for Sea Condition 2 (in all cases). In Sea Condition 1, although the results are generally better, no significant results are achieved. Interestingly, there are some cases in which 1−loop slightly outperforms HeuristicDivide, even though HeuristicDivide (theoretically) does not divide a cycle into smaller cycles unless it improves the worst idleness. The reason for that lies in the fact that deciding whether to break a big cycle into smaller cycles is done using an estimation of the cost of traveling along an edge, by averaging across simulations of ships patrolling along the edges of arbitrary paths, which are usually different then the paths found by HeuristicDivide's solution. For instance, consider the case where the final solution requires a 180° turn towards the next point. The physics of the ship movement dictates the ship to travel in an arc, which makes the path to the next point longer than its estimate. We leave the incorporation of the cost of traveling along angular paths in HeuristicDivide to future work.

7 Conclusions

In this work we introduced a new class of strategies for the frequency-based multiagent patrol problem suitable for multiagent patrol in complex environmental conditions. In this new strategy class, which we call MultiPartition, a graph is divided into disjoint cycles in which a subteam of the agents travel coordinatedly such that the maximal time between visits to interest points is minimized. This strategy class is a generalization of existing strategies that either create one cyclic path throughout the entire graph (SingleCycle strategies) or divide the graph into k disjoint subgraphs (k being the number of agents), where each agent patrols in its own subgraph—the UniPartition strategy. We showed that finding an optimal strategy to the problem is $\mathcal{NP}$-Hard in the general case, and also intractable in a realistic simplified family of outerplanar graphs. We then introduced a heuristic algorithm that divides the outerplanar graph into disjoint cycles. We evaluated our heuristic algorithm in a custom-developed ship simulator that realistically models ship movement constraints such as engine force and drag, and reaction of the ship to environmental changes. Results indicate that this algorithm significantly improves the frequency of visits compared to other known patrol strategies.

This work opens up several directions for the future. First, it would be interesting to consider the problem of multiagent patrol in prioritized environments, i.e., where vertices of the graph should be visited with different frequencies. Second, we intend to add more learning methods for determining the cost of travel, especially in prioritized environments. From a larger

perspective, this work raises the challenge of finding effective heuristics for the MultiPartition problem on general graphs.

References

1. N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proc. of ICRA'08*, 2008.
2. M. Ahmadi and P. Stone. A multi-robot system for continuous area sweeping tasks. In *Proc. of ICRA'06*, 2006.
3. A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyr. Recent advances on multi-agent patrolling. *Lecture Notes in Computer Science*, 3171:474–483, 2004.
4. N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proc. of AAMAS'09*, 2009.
5. G. Chartrand and F. Harary. Planar permutation graphs. *Annales de l'institut Henri Poincare' (B) Probabilite's et Statistiques*, 3(4):433–438, 1967.
6. Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proc. of IAT'04*, 2004.
7. A. de Mier and M. Noy. On the maximum number of cycles in outerplanar and series-parallel graphs. *Electronic Notes in Discrete Mathematics*, 31:489–493, 2009.
8. Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Math and Artificial Intelligence journal (AMAI)*, 57(3—4):293—320, 2009.
9. Y. Elmaliach, A. Shiloni, and G.A. Kaminka. A realistic model of frequency-based multi-robot fence patrolling. In *Proc. of AAMAS'08*, pages 63–70, 2008.
10. A. Machado, G. Ramalho, J.D. Zucker, and A. Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *Proc. of MABS'02*, pages 155–170, 2003.
11. J.S. Marier, C. Besse, and B. Chaib-draa. Solving the continuous time multiagent patrol problem. In *Proc. of ICRA'10*, 2010.