2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Workshops

Multi-A(ge)nt Graph Patrolling and Partitioning

Yotam Elor and Alfred M. Bruckstein.
Faculty of Computer Science and the Goldstein UAV and Satellite Center.
Technion, Haifa 32000, Israel.

***Abstract*—We introduce a novel multi agent patrolling algorithm inspired by the behavior of gas filled balloons. Very low capability ant-like agents are considered with the task of patrolling an unknown area modeled as a graph. While executing the proposed algorithm, the agents dynamically partition the graph between them using simple local interactions, every agent assuming the responsibility for patrolling his subgraph. Balanced graph partition is an emergent behavior due to the local interactions between the agents in the swarm. Extensive simulations on various graphs (environments) showed that the average time to reach a balanced partition is linear with the graph size. The simulations yielded a convincing argument for conjecturing that if the graph being patrolled contains a balanced partition, the agents will find it. However, we could not prove this. Nevertheless, we have proved that if a balanced partition is reached, the maximum time lag between two successive visits to any vertex using the proposed strategy is at most twice the optimal so the patrol quality is at least half the optimal. In case of weighted graphs the patrol quality is at least $\frac{1}{2}l_{min}/l_{max}$ of the optimal where l_{max} (l_{min}) is the longest (shortest) edge in the graph.**

***Index Terms*—Patrol, Graph Partition, Emergent Behavior, Nature-Inspired Paradigms, Swarm Intelligence.**

I. Introduction

To patrol is to continuously travel through an area. The purpose of patrolling is to visit every point in the area as often as possible. "Informally, a good (patrol) strategy is one that minimizes the time lag between two passages to the same place and for all places"[1]. It is convenient to model the area being patrolled as an undirected graph. In this model, time is discrete and will be measured in cycles. Using the undirected graph model we can loosely define the patrolling task as continuously visiting all the vertices of a graph. The *idleness* of a vertex is defined as the time since the last visit to this vertex by an agent. In this work we will address the *worst idleness* evaluation criterion defined in [1], [2].

We present the Balloon DFS algorithm (BDFS) as a patrolling strategy. The algorithm addresses a simplified version of the problem in which all edge lengths are assumed equal (an expansion to weighted graphs is described in Section VII). When k agents are patrolling a graph G using BDFS, they dynamically partition G into k connected subgraphs. Experiments show that in time the algorithm achieves a stable partition. We prove that the largest subgraph in any stable partition is of size at most $\frac{|G|}{k} + \frac{k}{2}$. Every agent patrols his subgraph using DFS hence the *worst idleness* is $2\frac{|G|}{k} + k$. In every time cycle every agent can visit at most one vertex so every multi-agent strategy (including the optimal) yields a *worst idleness* of at least $|G|/k$. Assuming $|G| \gg k$, the *worst idleness* of BDFS is about $2|G|/k$ which is at most twice the optimal.

Due to space limitations the proofs of the lemmas and some experimental results are omitted. They can be found in our readily available TR[3].

Some previously suggested partition based strategies work in two stages[4], [5]. In the first stage, the graph is partitioned between the agents and in the second the agents are sent to patrol the graph. The first stage complexity is high and requires some kind of “coordinator agent” who knows the graph in advance. In contrast, in our approach, the graph partitioning is executed "on the run", there is no need to a-priory know the graph and there is no need for any preprocessing before starting the patrol. Furthermore, if the graph changes during the run, or some agents break down, the agents automatically repartition the graph without any external intervention. In case of an agent break down the malfunctioning agent domain will be conquered and patrolled by other agents. On the downside, BDFS cover time is large and reaching a good partition may take a long time.

The algorithm can be classified as a local search algorithm. The agents continuously change the graph partition toward (hopefully) better configurations. Under the heavy restrictions of the lack of global information and the low communication and memory abilities of the agents a local search algorithm is a natural choice.

II. Previous Work

Given a graph and single agent to patrol it we can find a minimal length closed path which covers all vertices of the graph. Finding that kind of path is the well known traveling sales person problem (TSP). Chevaleyre *et al* have shown that an optimal single agent strategy would be to follow the best TSP circuit[4]. Considering a multi agent scenario, we can make all agents follow the same cycle while maintaining a constant gap between them or alternately partition the graph into subgraphs, each subgraph being patrolled by an agent performing an optimal circuit in it.

The scenario where the agents can sense the *idleness* of all the vertices of the graph lead to several *heuristic* approaches[1], [2]. Two different assumptions were made: either each agent can sense only the vertices’ *idleness* relative to it own visits or each agent sense the vertices’ *idleness* with respect to all agents. Note that in the second case the *idleness* of the vertices serves as a communication method. Among other strategies considered, a straightforward one was to choose the shortest route to the vertex with the highest

This research was supported by the Technion Goldstein UAV and Satellite Center and by the European Community’s FP7-FET program, SMALL project.

978-0-7695-3801-3/09 $26.00 © 2009 IEEE
DOI 10.1109/WI-IAT.2009.125

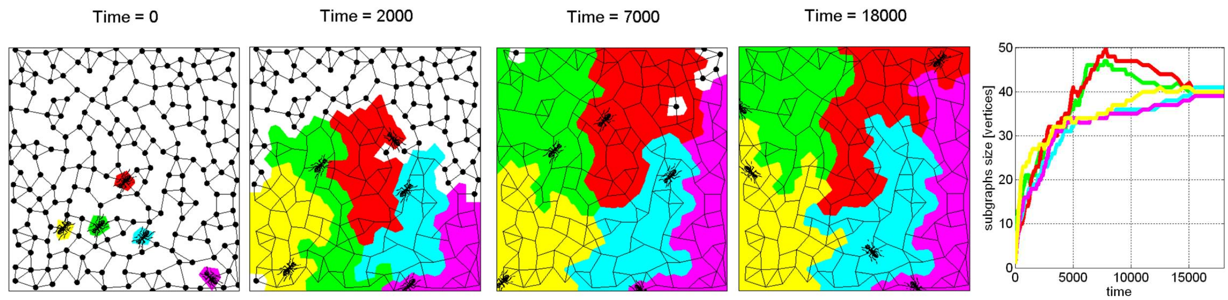


Figure 1: A typical run of BDFS on a random spatial graph with 5 agents. The different colors represent areas controlled by different agents. The graph was created by randomizing locations for 201 vertices on the plane and then connecting every vertex to its three closets neighbors. The number of vertices controlled by every agent as a function of time is presented in the rightmost figure. The resulting configuration after about $15,000$ time cycles is balanced.

idleness. A distributed sensing variation proposed by Wagner *et al* is to choose the vertex with the highest *idleness* only from the adjacent vertices as inspired by ants[6]. Yanovski *et al* have suggested another distributed sensing variation in which the agent marks the graph edges with pheromone marks (instead of the vertices) which leads to patrolling the graph using an Euler cycle[7].

Recently, Ahmadi and Stone[8] have proposed a patrolling strategy based on dynamic graph partition algorithm in which the agents exchange their boundary vertices using negotiating process. The algorithm they propose requires direct communication and computationally stronger agents. Furthermore, they do not have guarantees on the quality of the emergent partition.

In the heuristic algorithm ANTS proposed by Comellas and Sapena[9] some agents are working together to color a graph in k colors such that the weight of the cut between the colors is minimal. The subgraphs created using this approach are not connected. Another approach is constructing a "vertex-market"[2], [10]**.** Each agent owns some vertices and it is his responsibility to patrol his vertices. Agents can trade vertices in the market, for example an agent can trade a far vertex with a closer one and by that decrease its route length. In this scheme, a vertex can belong to several agent routes.

In the bubble scheme[11] k vertices are chosen as seeds and the seeds expand BFS-style (or "diffuse") to create a k-partition of the graph. Afterward, k new seed vertices are chosen based on the resulting partition and the process is repeated until the seeds do not change their location. Meyerhenke *et al* have proved convergence for a variant of the algorithm[12], however with no analytic guarantees on the partition quality.

Some recent works have suggested a new evaluation criteria for the patrolling problem[5], [13], [14]. Assuming some intruder model the new evaluation criterion is, roughly speaking, the ability to stop the intruder. The main theme of these patrolling strategies is to use probabilistic algorithms in order to avoid static patrolling patterns which can be exploited by intruders. However, stopping intruders is only one aspect of patrolling. For applications such as cleaning, maintenance, surveillance or guarding against weak intruders, the *worst idleness* criterion remains very important. Hence in this work, our goal is to minimize *worst idleness.*

III. High-Level description of the Algorithm and the Physical Intuition

Our algorithm is inspired from gas filled rubber balloons. Consider a space volume V with N balloons inside it. The balloons are all filled with the same quantity of gas and they are infinitely elastic. Given enough time, the system will converge to a balanced state in which the balloons fill the whole space V and the pressure inside the balloons is equal. Since the amount of gas in every balloon is the same, the balloons will have a volume of V/N each. The algorithm we propose mimics the balloons' behavior. Every agent controls a connected part of the graph. There are some border edges i.e. edges who connect vertices belonging to different agents. The agents sense those border edges and apply some "artificial pressure" over them - similarly to the pressure the gas molecules apply on the balloon surface. When the pressure applied by an agent is higher than the pressure applied from the other side, the agent can conquer the vertex on the other side. Since agents who control relatively small area hit the border edges more often than agents who control larger areas, they apply more pressure and the process described will hopefully converge to a balanced partition of the graph.

In physical balloon systems the space is continuous hence there is always a perfectly balanced partition of the space. In our system $|G|/k$ is generally not an integer so such a partition does not always exist. The most balanced partition is a partition in which some agents control subgraphs of size $\lfloor |G|/k \rfloor$ and some of size $\lceil |G|/k \rceil$. It is desired that such a partition will be stable i.e. no more conquests will happen. In order to achieve this we set the rule that an agent can conquer a vertex from another agent only if the size difference of the agents' subgraphs is (strictly) larger than 1. As a result, every partition in which the size difference between every two adjacent subgraphs is at most 1 is stable. We show that in a stable partition every subgraph is of size between $\frac{|G|}{k} - \frac{k}{2}$ and $\frac{|G|}{k} + \frac{k}{2}$ (see Lemma 3).

IV. The Algorithm

A. Model

Ant-like agents having limited capabilities are considered here. The agents are assumed to have little memory (a finite number of registers with $O\left(log_2 |G|\right)$ bits), they all execute the same algorithm and no direct communication is allowed between them. However, the agents can mark their neighborhood with pheromone stamps which can later be sensed. These markings are used as a primitive form of distributed memory and communication. The number of markings on every vertex or edge is fixed where the size of each marking is $O\left(log_2 |G|\right)$ bits.

For simplicity, we use a *strong synchronous* model. We assume that agents are operating in rounds. In every round the agents have slightly different timing phase so they never work simultaneously. In this model every passage between adjacent vertices takes one time cycle. Other operations like marking or sensing marks are assumed to be instantaneous. These *strong synchronous* assumptions can be relaxed later.

B. Multi Level DFS

Depth First Search (DFS) can be used in order to travel a graph. A variant of DFS called Multi Level DFS (MLDFS) was introduced by Wagner *et al*[15]. MLDFS is carried out by using marks on the graph edges and vertices. An important property of the MLDFS is its noise robustness. If at some point the agent loses its backtracking path he just starts a new search.

C. The Balloon Mechanism

In BDFS every agent controls a connected subgraph and is responsible for patrolling it. We introduce the balloon mechanism which eventually balances the graph partition. The balloon mechanism requires an additional mark on each edge: the pressure mark. The *pressure* on an edge is a *vector mark* i.e. a mark which gives opposite readings when being sensed from different sides of the edge. For example, consider the edge uv. Assuming $P(uv) = p$ i.e. the pressure of edge uv when sensed from vertex u is p; then $P(vu) = -p$ i.e. the pressure when sensed from vertex v is $-p$.

A *border edge* is an edge connecting two vertices belonging to different agents. Every time an agent enters a vertex for the first time in the current search, it senses the pressure of all border edges emanating from the vertex. Assuming the agent is on vertex u, let uv be a border edge and denote the sizes of the subgraphs containing u and v by $|G_u|$ and $|G_v|$ respectively. Upon sensing the pressure of edge uv the agent acts as follows:

1) $P\left(uv\right) \leq 0$ and $|G_u| \neq -P\left(uv\right) - 1$. In this case the agent announces his area size by setting $P\left(uv\right) \leftarrow |G_u|$.
2) $P\left(uv\right) \leq 0$ and $|G_u| = -P\left(uv\right) - 1$ implies with high probability that $|G_u| = |G_v| - 1$ so the agent will set $P\left(uv\right) \leftarrow 0$.
3) $P\left(uv\right) > 0$ implies with high probability that $|G_u| \leq |G_v| - 2$ so the agent will conquer vertex v with probability r.

We can show that most of the time the agent knows what his subgraph size is (see Lemma 1). A high-level pseudo code of BDFS can be found in Algorithm 1.

Algorithm 1: Balloon DFS

```
/* the agent is on vertex u */
while exist an unvisited border edge uv do
    mark uv as visited
    if P(uv) > 0 then
        with probability r conquer vertex v and go to v.
        (return)
    else
        if AreaSize = -P(uv) - 1 then
            P(uv) <- 0
        else
            P(uv) <- AreaSize
while exist an unvisited non-border edge uv do
    mark uv as visited
    if v is unvisited then
        go to v (return)
if there is a backtrack edge then
    backtrack (return)
else
    start new BDFS (return)
```

V. Analysis

The proposed algorithm behavior is analyzed by defining system configurations and deriving the transition probabilities between them. A full description of the system includes the areas controlled by agents, the agents locations, the agents routes and the pressure over border edges. Instead of using the full description, we will use a partial one including only the areas controlled by agents. Formally, a system configuration c is a mapping of vertices to markings, $c:\ v \in V \rightarrow \{id_0, ..., id_k\}$.

A. Observations

We first introduce some notations. The system comprised k agents patrolling undirected graph G with $|G|$ vertices. $G_u(t)$ is the subgraph controlled by agent u at time t and G_u^c is the subgraph controlled by agent u in configuration c. This notation implies that vertex $u \in G_u$. G_u is a connected subgraph of G. All the vertices of G_u are marked by the pheromone mark of agent u denoted by id_u. However it may happen during the execution of the algorithm that some vertices are marked by id_u but do not belong to G_u, after a *balloon explosion* which is discussed in section VI. Time periods are denoted by $[T_0, T_1]$ where $t \in [T_0, T_1]$ implies that $T_0 \leq t < T_1$. We will say that G_u is *stable* in a time period $[T_0, T_1]$ if for every two time cycles $t_0, t_1 \in [T_0, T_1]$, $G_u(t_0) = G_u(t_1)$. A configuration is *stable* in a time period if all the subgraphs are *stable* in that time period. It is convenient to define $\Delta u \triangleq 2|G_u| - 1$ which is agent u's route length. The following lemma describes the agent behavior when patrolling his domain.

Lemma 1: When agent u patrols G_u which is *stable* in $[T_0, T_1]$: (1) Agent u will start a new DFS search at time $t_0 \in [T_0, T_0 + \Delta u]$. (2) In $[t_0, T_1]$, agent u will perform sequential DFS searches using exactly the same route according to the MLDFS algorithm.

In order to calculate the transition probability between configurations, we derive the conquest probability over a border edge. We will abbreviate the sentence "agent u applies pressure on all border edges emanating from vertex u" to "u-press". According to the algorithm, after a u-press $P(uv) \geq 0$ and after a v-press $P(uv) \leq 0$. A conquest is possible when upon pressing a *border edge* the pressure is strictly positive. Consider a border edge uv. Upon a u-press, if the last press of edge uv was a v-press then $P(uv) \leq 0$ and agent u can not conquer vertex v. Denoting the event that agent u presses an edge uv and the last press of the edge was a u-press by "double u press". So a double press is a necessary condition for a conquest. However it is not sufficient as the next lemma shows.

Lemma 2: Given a system in configuration c which is stable in $[T_0, T_1]$ and a border edge uv where $|G_u^c| \leq |G_v^c|$ then the probabilities of a conquest in any time period of length $\Delta u \Delta v$ in $[T_0 + \Delta v, T_1]$ are:

$$p\,(\text{agent u conquers vertex v}) = \tag{1}$$
$$\begin{cases} (\Delta v - \Delta u)\, r & |G_u| \leq |G_v| - 2 \\ 0 & else \end{cases}$$
$$p\,(\text{agent v conquers vertex u}) = 0 \tag{2}$$

The time period $T_0' = [T_0, T_0 + \Delta v]$ is right after G_u or G_v have changed so it is possible that one of the agents has not yet found a stable DFS route. However, there is at most one v-press in that time period and if G_u and G_v are about the same size there are at most two u-presses. So the conquest probability in this time period can be bounded from above by:

$$\begin{aligned} p\,(\text{agent u conquers vertex v in } T_0') &\leq 2r \\ p\,(\text{agent v conquers vertex u in } T_0') &\leq r \end{aligned} \tag{3}$$

Since r is small we will neglect the probability of a conquest in this time period. We will later refer to a conquest in this time period as an unjustified conquest since it can happen whether $|G_u| \leq |G_v| - 2$ or not.

B. Balanced Configurations

A balanced configuration is a configuration in which for all border edges uv, $||G_u| - |G_v|| \leq 1$ and all vertices are "controlled" by agents. The next lemma shows that the size of any subgraph in a balanced configuration is bounded.

Lemma 3: For any balanced configuration b, the size of every subgraph G_i^b is bounded by $\frac{|G|}{k} - \frac{k}{2} \leq |G_i^b| \leq \frac{|G|}{k} + \frac{k}{2}$.

After defining a balanced configuration we must ask "which graphs enable balanced configurations?"[1]. To the best of our knowledge this variation of the graph partition problem have never been asked. A work by Gyori[16] implies that every k-connected graph can be partitioned into k balanced subgraphs. Another sufficient condition is the existence of a Hamilton path i.e. every graph that contains a Hamilton path contains a balanced configuration for any k. This condition implies that grids contain a balanced configuration.

[1]The exact question we ask is "which graphs can be partitioned into k connected subgraphs such as the size difference between any two adjacent subgraphs is at most 1?"

C. The Convergence Conjecture

We believe that if balanced configurations exist, the system will eventually converge to one of them. However, we could neither prove this nor find a counter example. One way to prove convergence is to show the following:

Conjecture 4: If a balanced configuration exists, from every configuration there is a probability $p \geq \epsilon > 0$ that the system will reach a balanced configuration within a finite time.

Lemma 5: When the system is in a balanced configuration there is a probability $p \geq \epsilon > 0$ that the system will remain in that configuration indefinitely.

We have proved Lemma 5. However, we could not prove Conjecture 4. Conjecture 4 is hard to prove because system transitions that lead to balanced configurations may be highly complex as shown in the next section.

VI. Discussion and Simulations

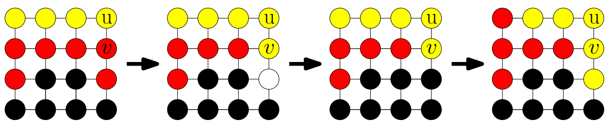
The quality of the partition can be measured using an "entropy" defined as:

$$E\,(c) \;=\; \begin{cases} -\sum |G_i^c| \cdot log|G_i^c| & \bigcup G_i^c = G \\ -\infty & else \end{cases}$$

The entropy is finite only if the subgraphs controlled by the agents cover the graph. In our case, a more balanced partition yields higher entropy.

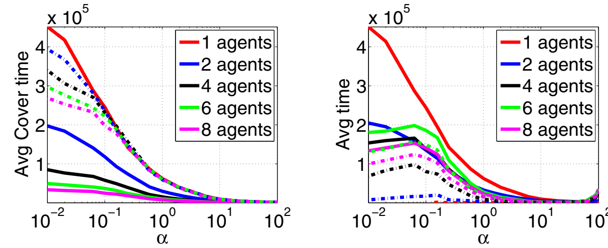
Let uv be a border edge in configuration c where $u \in G_u^c$, $v \in G_v^c$, $\bigcup G_i^c = G$ and $|G_u^c| \leq |G_v^c| - 2$. Assuming agent u conquers vertex v, let c' be the resulting configuration with $G_u^{c'}$ and $G_v^{c'}$ as the resulting subgraphs. If v is not a cut vertex of G_v^c then the conquest yields a better partition and higher entropy. However, if v is a cut vertex of G_v^c then $G_v^{c'}$ is the component of $G_v^c \backslash v$ agent v happens to be in during the conquest. We call this event a *balloon explosion*. Right after a *balloon explosion* there are some vertices controlled by no one so $E\,(c') = -\infty$. These empty vertices will shortly be conquered and the entropy will be finite again. However, the resulting entropy might be lower then the entropy prior to the uv conquest. Note that *Balloon explosions* are due to the discrete space we work in and would not occur in the physical balloons model described in section III.

Since any conquest which is a not a *balloon explosion* yields a better partition, *balloon explosions* might be considered destructive. It would be possible to change the algorithm so no *balloon explosions* will occur. However, *balloon explosions* are essential in order to escape deadlocks. An example for such a deadlock can be found in Figure 2. Several sequential configurations of a system in which three agents patrol a 4×4 grid are described in the figure. The different colors of vertices represent subgraphs controlled by different agents. In configuration c_1, the red and the black agents are controlling equal size subgraphs hence the conquest probability between them is zero. Any conquest of the yellow agent will result in a *balloon explosion*. In case we limit the agents to perform only conquests which does not cause *balloon explosions* the system



(c_1) E $\simeq -27$ (c_2) E $= -\infty$ (c_3) E $\simeq -27.2$ (c_4) E $\simeq -26.8$

Figure 2: An example of a deadlock on a 4×4 grid resolved by a *balloon explosion*.



(a) The solid lines represent the cover time and the dashed lines are the normalized cover time.

(b) The solid lines represent the convergence time and the dashed lines are the stabilization time.

Figure 3: Simulation results on random Hamiltonian graphs with varying edge-density and 101 vertices.

is in a deadlock and will never reach an optimal partition. A possible scenario leading to a better partition, when *balloon explosions* are allowed, is described in the figure. Note that the entropy of configuration c_3, which is after the vertices left empty by the *balloon explosion* were recaptured, is lower than the entropy prior to the *balloon explosion*. So c_3 is less balanced than c_1, however c_3 is better configuration since its closer to a stable one as can be seen in c_4. So sometimes less balanced configurations are actually better than more balanced ones. That phenomena makes proving conjecture 4 hard since we could not "measure" which configurations are closer to being balanced than others.

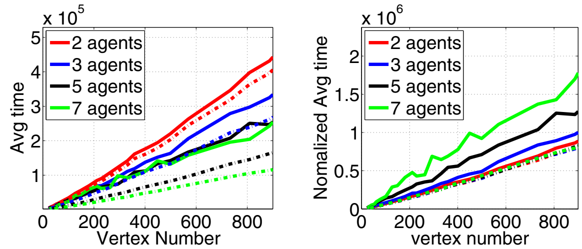
The convergence process can be divided into two phases: The cover phase starts when the algorithm starts and ends when the graph is covered. The stabilization phase starts when the graph is covered and ends when a stable configuration is reached. Respectively, the cover time is the time needed to cover the graph and the convergence time is the time to reach a balanced configuration. The experiments were done over grids and random Hamiltonian graphs. As mentioned before, assuming Conjecture 4 holds, the algorithm always converges to a balanced configuration on such graphs. A total of 42 grids were tested with size ranges between 4×5 to 15×16. The experimental results for the grids can be found in our TR[3]. A random Hamiltonian graph $H_n(p)$ is created by first constructing an n-cycle $v_1 v_2 \ldots v_n$ and then drawing additional edges at random with probability p for each possible edge. n is the number of vertices and the expected number of edges is $n + p\frac{n(n-3)}{2}$. The algorithm convergence time is influenced by the number of chords in the cycle. By setting $p \triangleq \alpha/n$ the number of chords is $\simeq \alpha n/2$ i.e. proportional to n and α. In order to avoid graphs where n/k is an integer, a primal number of vertices was chosen. The cover and convergence time of every experiment configuration was averaged over 500 runs. The algorithm converged in all our experiments - supporting the convergence conjecture.

Before performing simulations, the value of r (the conquest probability) must be set. r should be small enough to keep the probability of unjustified conquests low (see equation 3). On the other hand, choosing r too small will increase the convergence time because conquests will become rare. In our experiments $r = 0.01$ worked well.

Simulation results on random Hamiltonian graphs are presented in Figures 3 and 4. Observe Figure 3(a) for the average cover time as a function of the graph density. The solid lines represent the cover time and the dashed lines are the normalized cover time defined as the cover time multiplied by the number of agents. The cover time is decreasing with edge density. It is clear since the denser the graph there are more border edges and more conquest opportunities. For dense enough graphs the expansion rate of the area controlled by the agents is proportional to the number of agents. So we would expect the cover time to be proportional to $1/k$. The simulations revealed that for $\alpha > 10^{-0.8}$ our expectation is correct as the normalized cover times overlap. The very sparse graphs considered ($\alpha < 10^{-0.8}$) are rings with very few chords. In case of a single agent on a ring there are only two *border edges* no matter how big the subgraph controlled by the agent is. Furthermore, the path traveled by the agent when he controls a large subgraph is long hence the cover time is large. In case of k agents, there are about $2k$ *border edges* and the path traveled by each agent is relatively small. So using k agents instead of one improves the cover time with a factor larger than k. Therefore, experimentally, not only the absolute cover time improved when more agents were used but also the normalized cover time.

The average convergence and stabilization times as a function of the graph density are presented in Figure 3(b). We expected the convergence time to decrease when more edges are added to the graph since there will be more conquest opportunities and more balanced configurations. Simulations revealed that our expectation holds but for the two extremes of very sparse ($\alpha < 0.1$) and very dense ($\alpha > 50$) graphs. In the sparse extreme, the convergence time of graphs with $\alpha \simeq 0.1$ is larger than the convergence time of sparser graphs even though the cover time is smaller. The graphs with $\alpha < 0.1$ are rings with few chords so after they are covered there will be almost no *balloon explosions* delaying the stabilization. When α is increased more chords are added to the graph. The chords help to decrease the cover time but on the other hand they allow more *balloon explosions* during the stabilization phase resulting in longer stabilization phase and higher convergence time. On the other extreme, in very dense graphs there are many border edges so the probabilities of unjustified conquests can not be neglected. Hence unjustified conquests occur, and have a negative effect on the stabilization time.

Observe Figure 4(a) for the average convergence and cover times on random Hamiltonian graphs with $\alpha = 1$. The convergence and cover times are linear with the graph size. As



(a) (b)

Figure 4: Simulation results on random Hamiltonian graphs with varying size. The solid lines represent the convergence time and the dashed lines are the cover time.

expected, they are decreasing when more agents are added to the system. The convergence and cover time multiplied by the number of agents are presented in Figure 4(b). The normalized cover times overlap so the average cover time is $\Theta(n/k)$. However, the convergence time does not decrease linearly with k. When more agents are added the system, the stabilization phase is getting longer hence enlarging the convergence time. Formally, the convergence time is $\Theta(n/f(k))$ where f is a monotonic non-decreasing function.

VII. Weighted Graphs

In this section we will show how BDFS can be used to patrol weighted graphs. Let l_{max} (l_{min}) be the length of the longest (shortest) edge in the graph. The main idea, given a weighted graph G, is to convert it to an unweighted graph G' by prolonging all edges to l_{max}.

Prolonging all edges is easy when every agent knows l_{max}. It can be achieved in the following manner: Let the time required to traverse l_{max} be t_{max}. Every agent starts an internal clock whenever starting to traverse an edge. Upon reaching the other side of the edge, the agent will wait until time t_{max} and only then continue the algorithm. We do not address the problem of obtaining l_{max} and dispersing it between the agents. It can be easily done in preprocessing or "on the run".

Let G be a weighted graph containing n vertices and let G' be the graph obtained by prolonging all edges of G to l_{max}. The *worst idleness* of any k-agent patrolling strategy on G is at least $n \cdot l_{min}/k$ (when the graph contains a Hamilton cycle in which all edges length is l_{min}). The *worst idleness* of BDFS on G' is $l_{max} \cdot (2n/k + k)$. Assuming $n \gg k$,

$$\frac{worst\ idleness\,(BDFS)}{worst\ idleness\,(OPT)} \quad \leq \quad 2\frac{l_{max}}{l_{min}}$$

According to the equation above, the proposed expansion of BDFS to weighted graphs is efficient if the ratio l_{max}/l_{min} is low i.e. edges lengths do not vary too much. In case the graph is a result of a skeletonization process[17] this is frequently the case.

VIII. Conclusion

A novel graph patrolling algorithm was introduced. A group of k agents performing the algorithm partition the graph between them using simple local interactions. Graph partition into connected and size balanced components is an emergent behavior of the agent swarm. We have shown that in a balanced partition every subgraph is of size less than $\frac{|G|}{k} + \frac{k}{2}$. As a result, the time lag between two successive visits to any vertex is at most twice the optimal so the patrol quality is at least half the optimal. In case of a weighted graph we have shown that the algorithm achieves a competitive ratio of $2l_{max}/l_{min}$ when compared to the optimal patrol strategy where l_{max} (l_{min}) is the longest (shortest) edge in the graph. Simulations performed over grids and random Hamiltonian graphs showed that the average cover time is $\Theta(|G|/k)$. The average time to reach a balanced partition was experimentally found to be $\Theta(|G|/f(k))$ where f is a monotonic non-decreasing function.

References

[1] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul, "Multi-agent patrolling: An empirical analysis of alternative architectures," in *MABS*, 2002, pp. 155–170.

[2] A. Almeida, G. Ramalho, H. Santana, P. A. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre, "Recent advances on multi-agent patrolling," in *SBIA*, 2004, pp. 474–483.

[3] Y. Elor and A. M. Bruckstein, "Multi-a(ge)nt graph patrolling and partitioning," Computer Science Department,Technion Haifa, Israel, Tech. Rep., 2008.

[4] Y. Chevaleyre, F. Sempe, and G. Ramalho, "A theoretical analysis of multi-agent patrolling strategies," in *AAMAS*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 1524–1525.

[5] T. Sak, J. Wainer, and S. K. Goldenstein, "Probabilistic multiagent patrolling," in *SBIA*, 2008, pp. 124–133.

[6] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein, "Efficiently searching a graph by a smell-oriented vertex process," *Annals of Mathematics and Artificial Intelligence*, vol. 24, no. 1-4, pp. 211–223, 1998.

[7] V. Yanovski, I. A. Wagner, and A. M. Bruckstein, "A distributed ant algorithm for efficiently patrolling a network," *Algorithmica*, vol. 37, no. 3, pp. 165–186, 2003.

[8] M. Ahmadi and P. Stone, "A multi-robot system for continuous area sweeping tasks," May 2006, pp. 1724–1729.

[9] F. Comellas and E. Sapena, "A multiagent algorithm for graph partitioning." in *EvoWorkshops*, ser. Lecture Notes in Computer Science, vol. 3907. Springer, 2006, pp. 279–285. [Online]. Available: http://dblp.uni-trier.de/db/conf/evoW/evoW2006.html#ComellasS06

[10] T. Menezes, P. Tedesco, and G. Ramalho, "Negotiator agents for the patrolling task." in *IBERAMIA-SBIA*, ser. Lecture Notes in Computer Science, J. S. Sichman, H. Coelho, and S. O. Rezende, Eds., vol. 4140. Springer, 2006, pp. 48–57.

[11] R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw, "Shape-optimized mesh partitioning and load balancing for parallel adaptive fem," *Parallel Computing*, vol. 26, no. 12, pp. 1555 – 1581, 2000, graph Partitioning and Parallel Computing.

[12] H. Meyerhenke, B. Monien, and T. Sauerwald, "A new diffusion-based multilevel algorithm for computing graph partitions of very high quality," April 2008, pp. 1–13.

[13] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus, "An efficient heuristic approach for security against multiple adversaries," in *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2007, pp. 1–8.

[14] N. Agmon, S. Kraus, and G. Kaminka, "Multi-robot perimeter patrol in adversarial settings," May 2008, pp. 2339–2345.

[15] I. Wagner, M. Lindenbaum, and A. Bruckstein, "Distributed covering by ant-robots using evaporating traces," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 5, pp. 918–933, Oct 1999.

[16] E. Gyori, "On division of graphs to connected subgraphs." in *Combinatorics (Proc. Fifth Hungarian Colloq., Keszthely, 1976)*, vol. 1, 1976, pp. 458 – 494.

[17] B. W. Stout, "Smart moves: Intelligent path-finding." *Game Developer*, October 1996.