Efficiently Patrolling Hamiltonian and Two-Connected Graphs*

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

January 7, 2009

Abstract

We present two probabilistic patrolling strategies for a single agent patrolling an undirected graph. A single ant-like agent with very low capabilities is considered, the agent have small memory and he can sense only its neighborhood. However, he may mark the graph vertices with pheromone stamps which can later be sensed. These markings are used as a primitive form of distributed memory. The first algorithm presented is designed to patrol Hamiltonian graphs. By executing the proposed algorithm, the agent finds a Hamilton cycle and follows it. By following the cycle, the agent performs an optimal patrol. The second algorithm we present aims to patrol graphs whose square is Hamiltonian. Using the algorithm, the agent finds a cycle of length at most $2|G|$ and follows it. 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.

1 Introduction and Related Work

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" as described by [Machado *et al.*, 2002]. It is convenient to model the area being patrolled as an undirected graph. In this model, time is discrete and is represented in cycles. The patrolling task is defined as continuously visit all the vertices of the graph. Formally, the *idleness* of a vertex is defined as the time since the last visit to this vertex by an agent. The most common evaluation criteria for a patrol strategy is the *worst idleness* defined as the highest *idleness* of any vertex in the graph, defined by [Almeida *et al.*, 2004].

*This research was supported by the Technion Goldstein UAV and Satellite Center.

Algorithm 1: **VAW**

1 go to the neighbor vertex with the lowest σ value (brake ties randomly)
2 mark current vertex σ with t
3 $t \leftarrow t + 1$

A single ant-like agent with very low capabilities is considered. The agent has little memory (a finite number of registers with $O(log_2 n)$ bits where n is an upper bound on the graph size) and he can sense only its neighborhood. However, the agent can mark his close surroundings with pheromone stamps which can later be sensed. These markings are used as a primitive form of distributed memory. In our work, the pheromone markings are very simple, including only one time stamp per vertex.

The first algorithm we propose is based on the Vertex-Ant-Walk algorithm (VAW. see Algorithm 1). VAW is a simple patrolling procedure in which at every time cycle the agent takes a step to the neighbor vertex with the highest *idleness*. We prove that an agent performing VAW will eventually follow a cycle which covers the graph. However, this end-cycle is not necessarily simple thus not optimal. It was shown by [Wagner and Bruckstein, 1999] that a Hamilton cycle is a possible end-cycle of the VAW process. However, VAW sometimes converge to non-Hamiltonian end-cycles (see Figure 7 in the work of [Wagner *et al.*, 1998]). We propose the probabilistic-VAW (PVAW) algorithm as an expansion of VAW. An agent performing PVAW acts most of the time according to VAW. However, using only the time stamps, the agent knows when his end-cycle is not Hamiltonian so he performs random steps in order to find a better end-cycle. We prove that an agent performing PVAW on a Hamiltonian graph will reach a Hamiltonian end-cycle and will continue to follow it indefinitely thus patrolling the graph optimally.

Using a Hamilton cycle to efficiently patrol a graph is not a new idea. Finding a Hamilton cycle in a graph is NP-hard. Many approximation algorithms have been proposed[Angluin and Valiant, 1977; Bollobás *et al.*, 1987; Broder *et al.*, 1991]. Given a random Hamiltonian graph, these algorithms almost surely returns a Hamilton cycle within an average polynomial time. However, the polynomial time is achieved by assuming some distribution of the input graphs. The cycle based patrolling algorithms suggested so far are composed of two stages: first, using some approximation algorithms to find a good patrolling cycle. Second, sending the agent to follow that cycle[Chevaleyre *et al.*, 2004]. The first stage is complex and requires to know the whole graph in advance. Furthermore, if the graph changes, both stages must be repeated. In PVAW there is not such a separation to stages. The Hamilton cycle is found "on the run" so no preliminary knowledge is needed and whenever the graph changes, the agent finds a new Hamilton cycle (if one exists) without any outside intervention.

A single agent can use an Euler cycle to patrol a graph. It was shown by [Yanovski *et al.*, 2003] that an ant-like agent using time stamps on the graph edges (instead of the vertices) finds an Euler cycle, if such a cycle exist, and

uses it to patrol the graph. However, in this case, the *worst idleness* can only be bounded by $|G|^2$.

The second algorithm we propose (PVAW2) is a simple expansion of PVAW designed to patrol graphs whose square is Hamiltonian. The square of G (denoted by G^2) is the graph on the vertices of G in which two vertices are adjacent if and only if they have distance of at most 2 in G. Note that many of the graphs being patrolled are two-connected and "the square of every two-connected graph is Hamiltonian"[Fleischner, 1974]. The PVAW2 algorithm can be intuitively understood by considering an agent who can sense its surrounding to a distance of two edges (instead of one) and can travel two edges in a single time step (again, instead of one). Such an agent performing PVAW on G, while choosing his next step from a 2-neighborhood, is practically performing PVAW on G^2. Since G^2 is Hamiltonian, the resulting end-cycle will necessarily be Hamiltonian. Consider a more reasonable agent model who can sense its surrounding to a distance of two edges but can only travel one edge in a time step. We prove that such an agent performing PWAV2 on a graph which its square is Hamiltonian will find an end-cycle of length of at most $2|G|$ and will continue to follow it indefinitely thus yielding an approximation ratio of 2 regarding the *worst idleness* achieved.

Achieving an approximation ratio of 2 on the *worst idleness* is trivial using DFS. However, DFS is more complex and requires more pheromone marks. DFS is less robust and special attention is required if the graph might change (see a resilience DFS algorithm by [Wagner *et al.*, 1999]). Furthermore, while DFS approximation ratio is always 2, PVAW2 usually yields a shorter cycle hence a better approximation ratio on average (see Figure 3(b)).

2 Patrolling Hamiltonian Graphs

PVAW is a single-agent patrolling algorithm designed to patrol Hamiltonian graphs. An agent performing PVAW operates most of the time according to VAW. However it sometimes performs random steps in order to escape non-Hamiltonian end-cycles. An informal description of PVAW is the following: During the patrol, using only the time stamps, the agent knows if the current vertex and the previous visited vertex were visited consecutively and in the same order in the previous time they were visited. If its true, the agent performs the regular VAW i.e. goes to the neighbor with the lowest time stamp. Otherwise, with a small probability p the agents goes to a random neighbor or makes a regular VAW step with probability $1-p$. When the agent follows a Hamilton cycle, the vertices are always visited in the same order so the agent continues to follow the cycle indefinitely. When the agent is not following a Hamilton cycle, there will be infinitely many events of a small probability to change the agent route. Hence the agent will eventually change its route.

PVAW is presented here as Algorithm 2. $\sigma(v)$ are the time stamps on the graph vertices; $0 < p < 1$ is a constant parameter; x is a random variable chosen uniformly in $[0, 1]$; σ_{mem} and $PrevDiff$ are variables in the agent's memory.

Algorithm 2: **Probabilistic VAW.**

```
   /* the agent is on vertex u                                          */
1  if PrevDiff ≠ 1 and x ≤ p then
2  |  go to a random neighbor of u
3  else
4  |  go to the neighbor of u with the lowest σ value (brake ties randomly)
   /* the agent is on vertex v                                          */
5  PrevDiff ← σ(v) − σ_mem
6  σ_mem ← σ(v)
7  σ(v) ← t
8  t ← t + 1
```

2.1 PVAW Convergence Proof

Let $v_1 ... v_n$ be G vertices. The system comprises the agent patrolling the graph and the graph itself. The system's degrees of freedom are the time stamps on the graph vertices, the agent location and the agent's internal variables. Observe that the agent location is on the vertex with the highest time stamp and the current time equals to the highest time stamp plus one. So the values $\sigma(v_i)$, σ_{mem} and $PrevDiff$ fully describes a system state.

Using the notation $\sigma(v_0) \triangleq \sigma_{mem}$, a "full" system configuration comprises the values of $\sigma(v_i)$ and $PrevDiff$. However, sometimes it will be more convenient to use another configuration representation - a "squeezed" one. Consider the following squeezed configuration representation: The graph vertices (including v_0) are sorted in a non-decreasing order of their time stamps values, formally $v_{l_0} v_{l_1} ... v_{l_n}$ where $\forall l_i$, $\sigma(v_{l_i}) \leq \sigma(v_{l_{i+1}})$. Additionally, let $\Delta_i \triangleq \sigma(v_{l_{i+1}}) - \sigma(v_{l_i})$. However, if $\Delta_i \geq 2$, the exact value of Δ_i does not effect the agent behavior, so in the squeezed representation Δ_i takes only three values: $\{0, 1, \geq 2\}$. Furthermore, an agent performing the algorithm distinguishes between only two states of $PrevDiff$: $PrevDiff = 1$ and $PrevDiff \neq 1$, hence in the squeezed representation $PrevDiff \in \{1, \neq 1\}$. Since there is a finite number of permutations of l_i and the squeezed Δ_i and $PrevDiff$ take a finite number of values, there is a finite number of squeezed configurations denoted by M.

Full configurations are denoted by $c_0, c_1, ...$ and squeezed configurations by $s_0 ... s_M$. Every full configuration can be mapped to a squeezed one by sorting the vertices by their markings and "squeezing" Δ_i and $PrevDiff$. Let $sq(c_i)$ be the squeezed configuration mapped to c_i. If c_2 is the configuration succeeding c_1 then $sq(c_2)$ succeeds $sq(c_1)$.

The agent marks one vertex in every time cycle so after the graph is covered every two vertices will be marked with different time stamps. When there are no two vertices with the same time stamp VAW becomes a deterministic traversal of the graph. From now on it is assumed that the graph is already covered so VAW is deterministic.

Given a system in configuration s_0, we set $x > p$ for the next M time cycles so

for the next M time cycles the agent acts according to VAW. Recall that VAW is deterministic so we can predict the next M system configurations, denote them by $s_1...s_M$. There are only M distinct configurations hence there are i, j such as $0 \leq i < j \leq M$ and $s_i = s_j$. Assuming we set $x > p$ indefinitely, the system chain of configurations will be of the form $s_0...s_{i-1}(s_i...s_{j-1})^{\infty}$. While the system carries out the configuration loop $s_i...s_{j-1}$, the agent follows the cycle $u_i...u_{j-1}$ where u_l is the vertex with the highest σ value in configuration s_l. $u_i...u_{j-1}$ is the resulting end-cycle of VAW when initiated from s_0.

Denoting by $r(s_0)$ the shortest configuration loop in the set $s_0...s_{M+1}$ (the configuration loops in $s_0...s_{M+1}$ are cyclic permutations of $r(s_0)$ and some of their powers). We will use $s_l \in r(s_0)$ to imply that configuration s_l is in the configuration loop and $u_l \in r(s_0)$ to imply that in configuration s_l the agent is on vertex u_l. Let $u_l, u_p \in r(s)$, we will use the notion $u_l = u_p$ to imply that in configurations s_l and s_p the agent is on the same vertex. The end-cycle is not necessarily simple i.e. $l \neq p$ does not imply $u_l \neq u_p$ (however it does imply $s_l \neq s_p$). Let $r(c) \triangleq r(sq(c))$.

Lemma 1. *$r(c)$ includes all the vertices of G.*

Proof. Assume on the contrary that there are some vertices which are not in $r(c)$. Let $V \subset G$ be the subset containing all those vertices. Since G is connected, there is a vertex $u \in r(c)$ with a neighbor in V. Let t_0 be the time the agent enters the end-cycle . The vertices of V are not visited after t_0 hence $\forall v \in V$, $\sigma(v) < t_0$. After the agent have followed the end-cycle for the first time, $\forall u \in r(c)$, $\sigma(u) \geq t_0$. The next time the agent will reach vertex u, he will go to the neighbor with the lowest σ value which is in V - a contradiction. □

Corollary 1. $n \leq |r(c)| \leq M$.

Lemma 2. *If $|r(c)| > n$, there is a pair (u_i, u_j) where $u_i, u_j \in r(c)$ such as:*

1. $i < j$.
2. $u_i = u_j$.
3. $\forall i < l < j$, $u_l \neq u_i$.
4. $u_{i+1} \neq u_{j+1}$

Proof. Let $r(c) = u_1, ..., u_k$. Since $k > n$, there is a pair (u_i, u_j) fulfilling conditions 1-3. Assume on the contrary that for any such a pair condition 4 is false i.e. $u_{i+1} = u_{j+1}$. Without losing generality assume that $i = 1$, then $r(c)$ is of the following form:

$$\begin{aligned} r(c) &= u_1 u_2 ... u_j u_{j+1}, ... \\ u_1 &= u_j \quad u_2 = u_{j+1} \end{aligned}$$

The pair (u_2, u_{j+1}) fulfills conditions 1-3 so we can apply the assumption that condition 4 is false and write:

$$r(c) = u_1 u_2 u_3 ... u_j u_{j+1} u_{j+2} ...$$
$$u_1 = u_j,\ u_2 = u_{j+1},\ u_3 = u_{j+2}$$

The process can be repeated until for some vertex u_p, $u_p = u_j$. Then $r(c)$ is of the form $(u_1 ... u_{p-1})^m$ for $m > 1$, in contradiction to the definition of $r(c)$ as the shortest configuration loop (since $u_1 ... u_{p-1}$ is a shorter loop).

□

It is convenient to define $\sigma_{old}(v)$ as the second to last time stamp assigned to vertex v i.e. upon marking vertex v with a time stamp, the previous value is written to $\sigma_{old}(v)$. In case the agent have just taken a step from u to v but have not marked $\sigma(v)$ yet, the new $PrefDiff$ value is given by $\sigma(v) - \sigma_{old}(u)$. After the agent have marked v, $PrefDiff$ is given by $\sigma_{old}(v) - \sigma_{old}(u)$. In each time cycle the agent marks one vertex, so any two markings of any two different vertices (σ or σ_{old}) are different.

Lemma 3. *From any configuration c where $|r(c)| > n$, within M time steps the system will reach a configuration in which $PrevDiff \neq 1$.*

Proof. Let (u_1, u_j) be a pair fulfilling the conditions of Lemma 2. So $r(c)$ is of the form $u_1 u_2 ... u_j u_{j+1} ...$ where $u_1 = u_j$ and $u_2 \neq u_{j+1}$. For simplicity, let the time the system is in configuration c_1 be $t = 1$. So the agent marks vertex u_1 at time $t = 1$ and u_2 at time $t = 2$ i.e. $\sigma(u_2) - \sigma(u_1) = 1$. At time $t = j$ the agent visits u_j so $\sigma_{old}(u_j) \leftarrow \sigma(u_j) = \sigma(u_1)$. Let $s' \in r(c)$ be the configuration at time $t = j + 1$. Since $u_{j+1} \neq u_2$, $\sigma(u_{j+1}) \neq \sigma(u_2)$. In s', the agent calculates:

$$\begin{aligned} PrevDiff^{s'} &\leftarrow \sigma(u_{j+1}) - \sigma_{old}(u_j) \\ &\neq \sigma(u_2) - \sigma(u_1) = 1 \end{aligned}$$

Assume on the contrary that in the next M configurations following c, $PrevDiff = 1$. In this case the agent acts according to VAW so we can predict the next M configurations denoted by $s_1 ... s_M$. As mentioned before, the configurations $s_i ... s_{j-1}$ form the configuration loop $r(c)$. So there is a configuration $s' \in r(c) \subseteq s_0 ... s_M$ such as $PrevDiff^{s'} \neq 1$, a contradiction.

□

Let $p_0 \triangleq \frac{p}{\Delta}$ where Δ is the maximum vertex degree.

Lemma 4. *Let $u_1 u_2 u_3 u_4$ be a simple path in G and c_1 a configuration such as:*

1. *The agent is on vertex u_1.*
2. *$\sigma^{c_1}(u_1) > \sigma^{c_1}_{old}(u_1) > \sigma^{c_1}(u_2) > \sigma^{c_1}(u_3)$.*
3. *$PrevDiff^{c_1} \neq 1$.*

Table 1: The walk r from Lemma 4.

Agent Location	σ (u_1)	σ_{old} (u_1)	σ (u_2)	σ_{old} (u_2)	σ (u_3)	σ_{old} (u_3)
u_1	1	0	-1	?	-2	?
u_2	1	0	2	-1	-2	?
u_3	1	0	2	-1	3	-2
u_2	1	0	4	2	3	-2
u_1	5	1	4	2	3	-2
u_2	5	1	6	4	3	-2

There is a probability of at least p_0^5 that the system will reach configuration c_2 within 5 time cycles. Configuration c_2 fulfills:

1. *The agent is on vertex u_2.*
2. $\sigma^{c_2}(u_2) > \sigma^{c_2}_{old}(u_2) > \sigma^{c_2}(u_3) > \sigma^{c_2}(u_4)$.
3. $PrevDiff^{c_2} \neq 1$.

Proof. Consider the walk $r = u_1u_2u_3u_2u_1u_2$ described in Table 1. During this walk, $PrevDiff \neq 1$, so every step probability is at least p_0 and the probability of the whole walk is at least p_0^5. Denote the configuration after the walk r by c_2. According to table 1, $\sigma^{c_2}(u_2) > \sigma^{c_2}_{old}(u_2) > \sigma^{c_2}(u_3)$ and $PrevDiff^{c_2} \neq 1$. u_3 have been visited after u_4 hence $\sigma^{c_2}(u_3) > \sigma^{c_2}(u_4)$ and c_2 fulfills all the conditions of the lemma.

□

The resulting configuration of Lemma 4 fulfills the preconditions of the lemma with the simple path $u_2u_3u_4u_5$ (where $u_5 \neq u_1$) so the lemma can be applied again with the path $u_2u_3u_4u_5$.

Definition 1 (*semi-final* configuration)**.** *A* semi-final *configuration is a configuration in which there is a vertex ordering $u_1...u_n$ such as $\forall i$, $\sigma(u_i) < \sigma(u_{i+1})$ and $u_1...u_n$ is a Hamilton cycle. The next two lemmas discuss* semi-final *configurations.*

Lemma 5. *From any configuration there is a probability $P \geq \epsilon > 0$ that the system will reach a* semi-final *configuration within a finite time T.*

Proof. The case $|r(c)| = n$ is trivial since any of the loop configurations is *semi-final* and the loop is reachable with a probability of at least $(1-p)^M$ within M time steps. Assuming $|r(c)| > n$, withing M time steps the system will reach a configuration in which $PrevDiff \neq 1$ (Lemma 3). Let c' be the first such configuration and denote the agent location in c' by u_1. Let $C = u_1u_2...u_n$ be a Hamilton cycle in G such as $\sigma(u_2) - \sigma_{old}(u_1) \neq 1$. Such a cycle exist since for any Hamilton cycle C_0, C_0^{-1} is also a Hamilton cycle. Consider the following scenario:

1. From configuration c, take steps until reaching c'.
2. Consider the walk $u_1u_2u_1$. $PrevDiff^{c'} \neq 1$ so the first step probability is at least p_0. After taking the step, $PrefDiff \leftarrow \sigma(u_2) - \sigma_{old}(u_1) \neq 1$ hence the second step probability is also at least p_0.
3. Go to the neighbor with the lowest time stamp (with probability at least $min\{p_0, 1-p\}$). Denote this neighbor by v. Since $v \neq u_2$ and $\sigma(v) < \sigma_{old}(u_1)$, upon reaching v, $PrevDiff \neq 1$. So there is a probability of at least p_0 to return to u_1. Consider the walk u_1vu_1 and denote the resulting configuration by c_1.
4. c_1 fulfills the preconditions of Lemma 4 with the path $u_1u_2u_3u_4$. So there is a configuration c_2 reachable within 5 steps and with probability p_0^5. c_2 fulfills the preconditions of Lemma 4 with the path $u_2u_3u_4u_5$ so the lemma can be applied again. We will apply the lemma $n-3$ times, every time adding the next vertex of the Hamilton cycle C. The last use of the lemma is on the path $u_{n-3}u_{n-2}u_{n-1}u_n$.
5. Consider the walk $u_{n-2}u_{n-1}u_n$ which is possible with probability p_0^2 according to the proof of Lemma 4.

All the vertices u_i where $1 \leq i \leq n-3$ were last visited in configuration c_i (stage 4). The vertices u_{n-2}, u_{n-1}, u_n were last visited in stage 5 in that order. So the resulting configuration after stage 5 is *semi-final* with the Hamilton cycle C. The described scenario probability is at least $\epsilon = p_0^{5n-9} > 0$. The scenario length is bounded from above by $M + 5n - 9$ hence $T \leq M + 5n - 9$.

□

Lemma 6. *Starting from any* semi-final *configuration c, after* $n+1$ *consecutive steps in which* $x > p$*, the agent will follow a Hamilton cycle indefinitely.*

Proof. The *final* configurations $s'_1, ..., s'_n$ are defined by:

$$\begin{aligned} & l_0^{s'_j} = 0 \\ \forall 1 \leq i \leq n, \quad & l_i^{s'_j} = (i-j-1) \, mod \, n + 1 \\ \forall 0 \leq i \leq n-1, \quad & \Delta_i^{s'_j} = 1 \\ & PrevDiff^{s'_j} = 1 \end{aligned}$$

In configuration s'_j the agent is on vertex u_j and the vertex with the lowest σ value is u_{j+1}. $u_1...u_n$ is a Hamilton cycle so there is an edge u_ju_{j+1}. $PrevDiff^{s'_j} = 1$, so independent of x, the agent will go to u_{j+1} and the resulting configuration is s'_{j+1}. Upon reaching one of the states $s'_1, ..., s'_n$, the system will follow this configuration loop indefinitely and the agent will follow the Hamilton cycle $u_1...u_n$.

Let the agent location at the *semi-final* configuration c be vertex u_k. After performing $n+1$ steps in which $x > p$ the resulting system configuration is the *final* configuration $s'_{(k+1) mod\, n}$ concluding the proof.

□

Theorem 1 (PVAW Convergence). *For any Hamiltonian graph G an agent performing PVAW on G will eventually patrol the graph using a Hamilton cycle.*

Proof. Assuming the agent starts the algorithm at time $t = 0$, let $P(t)$ be the probability that the agent will reach a Hamilton end-cycle by the time $t + T$. We are required to show that $\lim_{t\to\infty} P(t) = 1$.

From any configuration, with probability $P \geq \epsilon_1 = p_0^{5n-9}$, the system will reach a *semi-final* configuration within a finite time T (Lemma 5). Upon reaching the *semi-final* configuration, with probability $P \geq \epsilon_2 = (1-p)^{n+1}$, the agent will patrol the graph using a Hamilton cycle indefinitely (Lemma 6). We conclude that starting from any configuration, with probability $P \geq \epsilon_1 \cdot \epsilon_2 > 0$, after a finite time T the agent will patrol the graph using a Hamilton cycle indefinitely.

Let $\bar{P}(t) = 1 - P(t)$ i.e. $\bar{P}(t)$ is the probability that the agent will not reach a Hamilton end-cycle by the time $t + T$. $\bar{P}(t)$ can be bounded from above by $(1 - \epsilon_1\epsilon_2)^t$.

$$\lim_{t\to\infty} P(t) = 1 - \lim_{t\to\infty} \bar{P}(t) \geq 1 - \lim_{t\to\infty} (1 - \epsilon_1\epsilon_2)^t = 1$$

□

3 Patrolling Two-Connected Graphs

We denote by G^2 (the square of G) the graph on the vertices of G in which two vertices are adjacent if and only if they have distance of at most 2 in G. A slightly stronger agent enables the use of PVAW to efficiently patrol graphs whose square is Hamiltonian. The intuition behind the algorithm can be found in section 1.

In the patrolling problem it is usually assumed that an agent can travel only one edge per time cycle. So an agent who can sense to a distance of two edges, but can travel only a single edge per time cycle is considered here. This agent can perform the algorithm PVAW2 (presented here as Algorithm 3). Note that t in PVAW2 does not necessarily complies to real time because moving from vertex to vertex might take one or two time cycles. Whenever the agent "jumps" from u to v where $d(u, v) = 2$, the agent real path is u, w, v where w is a vertex connecting u and v. The d-neighborhood of u includes all the vertices of distance d or less from u and is denoted by $N_d(u)$.

3.1 PVAW2 Convergence Proof

The convergence proof of PWAV2 is based on PVAW convergence proof. We will define two systems and will show that they are similar in some sense. The

Algorithm 3: **Probabilistic VAW2.**

```
   /* the agent is on vertex u                                          */
1  if PrevDiff ≠ 1 and x ≤ p then
2  |  go to a random vertex v ∈ N2(u) (might take two time cycles)
3  else
4  |  go to the vertex v ∈ N2(u) with the lowest σ value (brake ties
   |  randomly. might take two time cycles)
   /* the agent is on vertex v                                          */
5  PrevDiff ← σ(v) − σ_mem
6  σ_mem ← σ(v)
7  σ(v) ← t
8  t ← t + 1
```

system A_1 is comprised of an agent performing PVAW2 on G and A_2 of an agent performing PVAW on G^2. A configuration, as defined in section 2.1, consists of the values $\sigma(v_i)$ and $PrevDiff$. Because G and G^2 have the same vertex set, any configuration can be ascribed to both A_1 and A_2. The next lemma shows that when a chain of system configurations is examined, these two systems are equivalent.

Lemma 7. *Given two configurations c_1 and c_2, the transition probability between the configurations is equal in A_1 and A_2.*

Proof. The crucial observation is that the 1-neighborhood of any vertex u in A_2 is identical to the 2-neighborhood of u in A_1. The agents are performing the same operations on the same vertices sets hence the transition probabilities between configurations are equal.

□

Theorem 2 (**PVAW2 Convergence).** *For any graph G such as G^2 is Hamiltonian, an agent performing PVAW2 on G will eventually patrol the graph using a cycle of length at most $2n$.*

Proof. According to Theorem 1, A_2 will eventually follow a configuration loop in which the agent performs a Hamilton cycle. So A_1 will also eventually follow such a configuration loop (Lemma 7). However, the circuit the agent performs in A_1 is of the form $u_1w_1u_2w_2...u_nw_n$ where $u_1..u_n$ is a Hamilton cycle in G^2 and w_i is the vertex connecting u_i to u_{i+1}. Since some u-vertices may be connected without requiring a w-vertex between them, the cycle used by the agent is of length of at most $2n$.

□

The optimal patrol strategy is finding a Hamilton cycle in G (if such a cycle exist) and following it. The optimal patrol yields a *worst idleness* of $n-1$. PVAW2 finds a cycle of length at most $2n$ hence the *worst idleness* achieved is

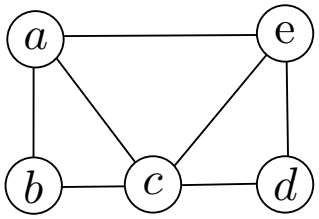


Figure 1: A simple convergence example.

at most $2n-1$. Hence PVAW2 yields an approximation ratio of 2 regarding the *worst idleness*.

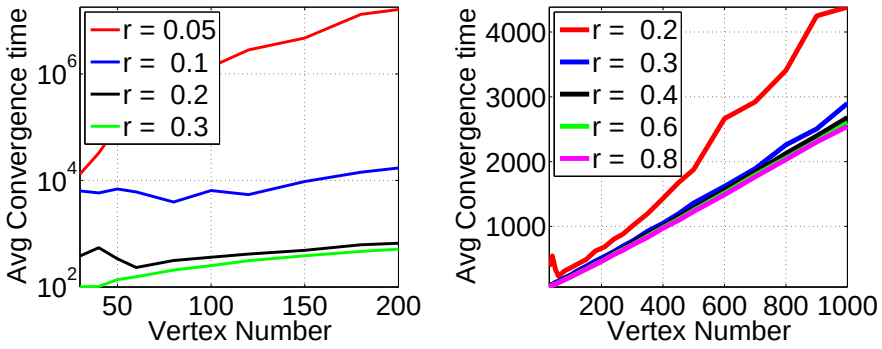
4 Discussion and Simulations

The convergence proof might give the impression that the scenario leading to convergence is highly improbable and as a result convergence takes a lot of time. However, simulations on random Hamiltonian graphs reveal that this is rarely the case. There are simpler and more probable scenarios leading to convergence. For example, consider the graph in Figure 1. A possible end-cycle of the VAW process is $r(c) = abcedc$. The vertex c is visited twice in each cycle so upon hitting c, $PrevDiff \neq 1$. Every time the agent reach vertex c there is a probability p that it will choose a neighbor randomly. If the agent will randomly go to either b or d the resulting configuration is *semi-final*. So every time the agent hits vertex c there is a probability of $p/2$ to reach a *semi-final* configuration. Upon reaching the *semi-final* configuration, there is a probability of $(1-p)^{n+1}$ to follow a Hamilton cycle indefinitely. These probabilities are much higher than the probabilities in the proof ($\epsilon_1$and ϵ_2).

Our experiments were done on random Hamiltonian graphs $H_n(r)$, where n is the number of vertices and r-the probability of additional edges beyond the basic cycle $u_1u_2...u_n$. Thus we start with an n-cycle and draw additional edges at random with probability r for each possible edge. The expected number of edges is $n + r\frac{n(n-3)}{2}$.

Since PVAW always finds a Hamilton cycle, the interesting question is how fast? The average time to find a Hamilton cycle is presented in Figure 2. Our simulations revealed that convergence is slower on low edge-density graphs. The lower the density, the longer the convergence time. However, above the density threshold of $r \simeq 0.2$, adding more edges does not improve the convergence time significantly. The explanation is simple, the denser the graph, there are more Hamilton cycles in it and more ways to improve the agent route so convergence is faster. The average convergence time for dense enough graphs was found to be about $2n$ i.e. linear with the graph size.

Recall that PVAW2 end-cycle is of length between n and $2n$. So when experimenting with PVAW2 we ask two questions: how fast the algorithm converges



(a) Low edge-density graphs.

(b) High edge-density graphs.

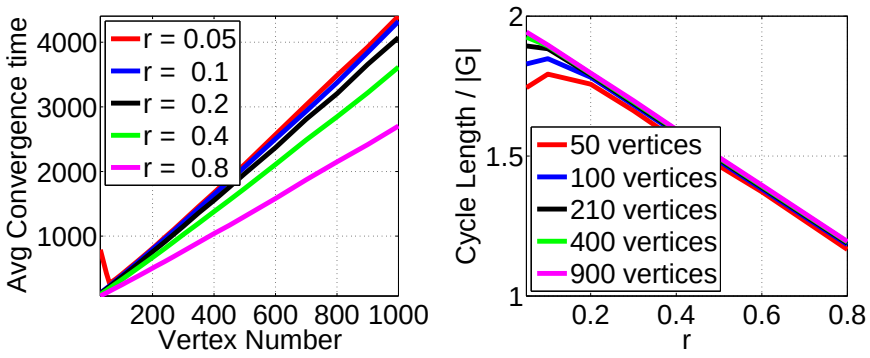
Figure 2: PVAW simulation results.

to an end-cycle? and what is the solution quality? The solution quality is the end-cycle length divided by the optimal end-cycle length. Our experiments were done on Hamiltonian graphs so the optimal end-cycle length is n. Observe Figure 3(b) for the average solution quality. For very sparse graph the end-cycle length is about $1.8n$. The denser the graph the solution quality is higher. Note that in the PVAW2 algorithm there is not any built-in mechanism preferring shorter end-cycles. When the agent takes a random step, it chooses uniformly from its 2-neighborhood. Consider the following variation of PVAW2: Whenever taking a random step, go to vertices in the 1-neighborhood with higher probability than the farther vertices. An agent performing this variation will probably find a shorter end-cycle however the convergence time might increase.

In contradiction to PVAW, PVAW2 converges in linear time even on very sparse graphs (observe Figure 3(a)). Since an agent performing PVAW2 on G is equivalent to an agent performing PVAW on G^2, even when G is sparse, G^2 is quite dense. Hence the convergence time of PVAW2 on sparse graphs is proportional to the convergence time of PVAW on dense graphs which is linear with the graph size.

5 Conclusion

In this paper we have presented two single-agent probabilistic patrolling algorithms. The first algorithm (PVAW) optimally patrols Hamiltonian graphs using a Hamilton cycle. We have proved that an agent performing PVAW finds Hamilton cycle and uses it to patrol the graph. Simulations on random Hamiltonian graphs showed that the average time to find such a cycle is about $2n$ for dense enough graphs. The second algorithm (PVAW2) aims to patrol graphs whose square is Hamiltonian. We have proved that an agent performing PVAW2 on such a graph finds a cycle of length at most $2n$ and uses it to patrol the graph.



(a) Average time to find an end-cycle.

(b) Average solution quality.

Figure 3: PVAW2 simulation results.

This strategy yields an approximation ratio of 2 to the optimal patrol strategy regarding the *worst idleness* achieved. Experiments on random Hamiltonian graphs showed that the average time to find such a cycle is about $3n$ and the length of the cycle found decreases with the graph edge-density, the denser the graph the shorter the cycle and the better the patrol.

References

[Almeida *et al.*, 2004] Alessandro Almeida, Geber Ramalho, Hugo Santana, Patricia Azevedo Tedesco, Talita Menezes, Vincent Corruble, and Yann Chevaleyre. Recent advances on multi-agent patrolling. In *SBIA*, pages 474–483, 2004.

[Angluin and Valiant, 1977] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. In *STOC*, pages 30–41, New York, NY, USA, 1977. ACM.

[Bollobás *et al.*, 1987] B. Bollobás, A. M. Frieze, and T. I. Fenner. An algorithm for finding hamilton paths and cycles in random graphs. *Combinatorica*, 7(4):327–341, 1987.

[Broder *et al.*, 1991] Andrei Z. Broder, Alan M. Frieze, and Eli Shamir. Finding hidden hamiltonian cycles. In *proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 182–189, 1991.

[Chevaleyre *et al.*, 2004] Yann Chevaleyre, Francois Sempe, and Geber Ramalho. A theoretical analysis of multi-agent patrolling strategies. In *AAMAS*, pages 1524–1525, Washington, DC, USA, 2004. IEEE Computer Society.

[Fleischner, 1974] Herbert Fleischner. The square of every two-connected graph is hamiltonian. *Journal of Combinatorial Theory*, 16(1):29–34, 1974.

[Machado *et al.*, 2002] Aydano Machado, Geber Ramalho, Jean-Daniel Zucker, and Alexis Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *MABS*, pages 155–170, 2002.

[Wagner and Bruckstein, 1999] I.A. Wagner and A.M. Bruckstein. Hamiltonian(t) - An Ant-Inspired Heuristic for Recognizing Hamiltonian Graphs. In *Proceedings of Congress on Evolutionary Computation (CEC99)*, Washington DC, July 6-9 1999. IEEE Press.

[Wagner *et al.*, 1998] Israel A. Wagner, Michael Lindenbaum, and Alfred M. Bruckstein. Efficiently searching a graph by a smell-oriented vertex process. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):211–223, 1998.

[Wagner *et al.*, 1999] I.A. Wagner, M. Lindenbaum, and A.M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *Robotics and Automation, IEEE Transactions on*, 15(5):918–933, Oct 1999.

[Yanovski *et al.*, 2003] Vladimir Yanovski, Israel A. Wagner, and Alfred M. Bruckstein. A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003.