Probabilistic Multiagent Patrolling

Tiago Sak, Jacques Wainer, and Siome Klein Goldenstein

Instituto de Computação, Universidade Estadual de Campinas,
CEP 13084-851, Campinas, SP - Brasil
{tiago.sak,wainer,siome}@ic.unicamp.br

Abstract. Patrolling refers to the act of walking around an area, with some regularity, in order to protect or supervise it. A group of agents is usually required to perform this task efficiently. Previous works in this field, using a metric that minimizes the period between visits to the same position, proposed static solutions that repeats a cycle over and over. But an efficient patrolling scheme requires unpredictability, so that the intruder cannot infer when the next visitation to a position will happen. This work presents various strategies to partition the sites among the agents, and to compute the visiting sequence. We evaluate these strategies using three metrics which approximates the probability of averting three types of intrusion - a random intruder, an intruder that waits until the guard leaves the site to initiate the attack, and an intruder that uses statistics to forecast how long the next visit to the site will be. We present the best strategies for each of these metrics, based on 500 simulations.

1 Introduction

Patrolling an environment can be seen as finding efficient ways of performing visits to all the important points of a given area. This task can be considered inherently multiagent-like since in most cases the process will be started in a distributed manner, by a group of agents. Research on multiagent patrolling, however, is not limited to patrolling real-world problem, but they can find applications on several domains, such as network security systems and games. In other words, patrolling can be useful in any domain characterized by the need of systematically visiting a set of predefined points. Additionally, it is important to keep the visit order secret, since an intruder could use it to plan a path that avoids being seen by the patrols.

The existing works in the area give priority to shorten the time spent between visits to all the regions in the environment, proposing methodologies without any kind of variation in the order which the points are visited. Although these methods achieve an efficient way of performing the visits, the order with which the nodes are visited can be easily deducted by any external intruder. By knowing the visit-order and the period between visits, an intruder can mount a successful intrusion. For example, if the intruder knows that a security guard visits a safe with a combination lock every 50 minutes regularly, he may successfully open the lock, even if the whole process takes longer than 50 minutes. By interrupting his attempt just before the guard arrives and hiding, and because his attempt does not leave a traces in the lock that can be recognized by the guard, the intruder will have time to try all lock combinations until he can open the safe. Of course if the intruder is trying to open the safe by cutting it with a welding torch, the interruption trick will not work. If cutting the safe takes 60 minutes, then a guard that returns every 50 minutes guarantees that the intrusion will be stop. But if cutting the safe takes 40 minutes, and the intruder can wait until the guard leaves the safe to start the attack, then a fixed period of 50 between visits will not detain the intrusion. Thus, depending on the type of intrusions, unpredictability of the period of the visits, or at least variability may be more important than shortening the period.

In this work, we will model the problem of patrolling as a problem of visiting vertices in a graph. The patrol will move from vertex to vertex, transversing the edges. Each edge has a value that corresponds to the time needed to traverse it. Each vertex is a place that needs to be observed for intruders, but we will assume that such observation are instantaneous. Each patrol will be called an "agent". The intruder will choose a vertex, and will stay some time at that location to achieve the intrusion or attack. If a patrol visits the vertex while the intruder is there, the attack has been averted. If not, the attack is successful and the patrolling task failed. Intruders do not move in the graph.

The multiagent patrolling task can be partitioned into two almost orthogonal decisions: if and how to divide the work, and how to select the visiting sequence for each agent. The first decision is whether all agents will patrol the whole graph or if each one will have a different subgraph to patrol. We call the second alternative the decision to *partition* the graph, and for the lack of better name, we call the alternative a *non-partitioning* decision. If the graph will be partitioned, then one has to decide on how to select the subgraphs that will be assigned to each agent. We call it the *partition algorithm.*

Independent of the partitioning decision, one has to select the sequence of vertex visitation for each agent. We call this the *sequencing algorithm.* In this paper we will only deal with *homogeneous* agents, that is, they all use the same sequencing algorithm. The combination of the partitioning decision, the partition algorithm (if needed) and the sequencing algorithm is known as a *strategy.*

This paper is organized as follows: the next section briefly describes previous works on multiagent patrolling; section 3 introduces our approach in details; section 4 presents an experimental evaluation; and, in section 5, we discuss the results and directions for the future.

2 Related Work

The use of multiples agents, acting cooperatively or not, to perform search of intruders or a patrolling tasks, have been studied by many researchers. [1] proposed the combination of map-learning and pursuit of invaders in a single problem, using a *greedy* policy that directs the pursuers at each instant to the location that maximize the probability of finding an invader. [2] studied the problem of generating near-optimal trajectories that will be followed by several agents to co-operatively search for targets in an environment with some a priori information about the target distribution. [3] investigated multiagent pursuit of targets that become observable for short time periods, this is used by each agent to estimate the next position of any intruder. [4] studied the use of stochastic rules to guide agents motions in order to perform the surveillance task. Their main interest was to define the rate at which the Markov chain converges to its steady state distribution. Thus they have not defined any evaluation criteria to precisely measure effectiveness of their strategies, and have not made any comparison with other possible architectures.

Very relevant to this work, is the work of Machado et al[5], which was followed by others [6,7,8,9]. These works also model the problem of patrolling as graph traversal. Their goal was to reduce the period between two consecutive visits to any vertex. In [5] they explore different alternatives for a local decision (by each agent) on the next vertex to visit, based on properties of the agents and communication abilities. [8] considered the use of reinforcement learning to discover plans for traversing the graph. All these are variations to the sequencing algorithm. [9] explores alternatives for the partitioning algorithm based on different alternative of negotiations protocols among the agents.

[6] uses the well-know *travelling salesman problem* (TSP) as a sequencing algorithm. Consider the problem of minimize the period between two consecutive visits to a vertex. This problem can be seen as finding the minimal cycle that contains all the vertexes in the graph, which is exactly the TSP definition. Although this is a *NP-hard* problem, there techniques to solve TSP can now determine the optimal result on very large instances of the problem. The standard TSP solution is designed for one agent traveling the graph. Two multiagent extensions were proposed: the first dispose the agents on the TSP-cycle, keeping an approximate distance between them. The second strategy partitions the graph into disjoint regions, each agent is assigned to patrol one region. Except for specific cases, the cyclic strategy achieved better result. As the expected, the TSP-based solution achieved the best results of all other alternatives for most cases.

3 The Probabilistic Patrolling Problem

3.1 Evaluation Criteria

We define three evaluation criteria for any solution to the problem. These criteria are approximations to detecting an attack by three different kind of intruder.

The **random intruder** will start an attack on a random node at a random time. The attack must last A to be successful. If within the time interval A, a agent visits the attacked vertex, the intrusion is averted.

The **waiting intruder** will wait (hidden) until the agent leaves a (random) vertex and start the attack then.

The **statistical intruder** will collect statistics on the period between visits to a random node and will initiate the attack whenever the data assures that it is safe to attack. The statistical intruder keeps a visit history, and computer the correlation ρ between the period of two consecutive visits to its goal node. Given that the last period to visit the node was x, the attacker will search the visit history to the node and find out the closest period to x in the record, and discover the following visit period y. If $y \times \rho > A$ the intruder will initiate the attack. ρ is a measure of how certain the intruder is that the next period will be y (because it was so in the past).

These intruders are in an increasing order of sophistication. The metrics are approximations to the probability of averting each of these intruders. PRI(A) is the probability of catching a random intruder with attack of length A. PWI(A) is the probability of catching a waiting intruder, and PSI(A) is the probability of catching a statistical intruder.

3.2 Sequencing Algorithms

We discuss two main alternatives to the sequencing algorithm. The random walk alternatives, have high variability and unpredictability, while the TSP-based solutions achieve shorter visiting periods, but have lower variability.

Random walk based: The intuitive idea of the *random walk* is to take successive steps, each one in a random direction. In graphs, random walks are discussed in [10]. There are theoretical results that with enough time every vertex in a graph will be reached, and there are lower and upper bounds to cover completely a graph. Since, every vertex will be visited in a random walk, one can consider using random-based solutions as the sequence algorithm. We propose two random based algorithms:

– **Local random algorithm:** The agent will choose randomly one vertex among all the adjacent nodes.
– **Global random algorithm:** The agent will choose randomly any vertex of the graph to be the next objective-node. To reach this vertex in case there is no direct edge between the current vertex and the objective-node, the agent will use the *shortest path* between them.

TSP-Based: As discussed above, a solution to the *travelling salesman problem* (TSP) on the graph is clearly a sequence that will minimize the period between visits to each node in the graph. But this solution has no variability or unpredictability, and thus will not stop some of the intruders discussed above. But given the TSP solution to a graph, there are alternatives to add variability to the sequence of visits.

– **Original TSP:** This is the sequence generated by the TSP solution of the graph. The TSP cycle is repeated continuously until the end of the task.
– **TSP with local visits:** The TSP cycle is covered over and over, but for each node visited, the agent decides randomly with probability $LV_{\%}$ if a *local*

visit will be performed or not. A *local visit* consists in visiting one neighbor of the current node, after this the agent returns to the original node and the next node in the cycle will be visited.
- **TSP with local changes:** This algorithm proposes that for each cycle the agent will perform a random local change in the TSP-cycle. Two nodes are chosen randomly and the order in which they appear in the TSP-cycle is exchanged. For example, if the original TSP-cycle was $\{v_1, v_2, v_3, v_4, v_5, v_6\}$, and the two nodes chosen were v_2 and v_5 then the new cycle would be: $\{v_1, v_5, v_3, v_4, v_2, v_6\}$.
- **TSP rank of solutions:** This algorithm is based on the fact that although there may be only one optimal solution, sub-optimal solutions to the TSP problem may also be efficient, and will provide variability. It is possible to some branch and bound TSP solvers to generate also other, sub-optimal solutions. Every time a feasible solution is found, it is stored in a priority queue. The TSP solver will return not only the optimal solution (which is the first element in the queue), but the K first elements of the heap, provided the cost of each solution is less than twice the cost of the optimal solution. The TSP rank of solutions algorithm will for each cycle, choose randomly one of the K solutions returned by the altered TSP solver.

3.3 Partitioning

Non-partitioning Cyclic Strategy: In this strategy all the agents will act with the same set of nodes. Although the vertices which the agents will have access are the same, they will act independently making their decisions without any influence from others agents. Nevertheless, it is very important to defined a way in which the agents will be distributed initially in the graph. Two alternatives are presented:

- **Random cycle:** in this alternative the agents will be distributed randomly through all the vertices of the graph, the only criteria followed was that one vertex can not be assigned to more than one agent.
- **Approximate equal distance cycle:** the second alternative distribute the agents equidistantly in the TSP-cycle, that is, from one agent to the next in the cycle there exists a constant distance, that will be repeated to all the agents. This constant distance is equal to the optimal TSP-solution size divided by the number of agents. In many cases, however, the perfect distribution of the agents will not be possible. Hence, this strategy finds the best, but not necessarily optimal solution. This is achieved by verifying all the possibilities and choosing the one which less distortion of the ideal equidistant distribution.

Partition algorithms: The problem of partitioning a graph is well known and important in a wide range of applications. There are, however, many possible approaches to perform this task, all of them based on different premises and goals. Three possibilities were explored in this work.

- **Multilevel graph partitioning:** This partitioning scheme gives priority to the size of the resulting partitions. Although, for the multiagent patrolling problem a fair partitioning of the graph is one where the distances travelled by each agent are equal, we approximate this requirement to determining subgraphs with approximately the same number of vertexes.
- **K-Means:** We use the K-means clustering algorithm [11] to partition the graph in K clusters based on the euclidean distance of each vertex to the prototype of each cluster (see section 4.1 for the discussion on "euclidean distance" of vertexes).
- **Agglomerative hierarchical clustering:** Agglomerative hierarchical clustering [11] is also a clustering algorithm, and in particular using the *single linkage* alternative for metric for the algorithm seems to closely correspond to the idea that close by vertexes should be joined into the same cluster, and that each agent should be responsible for one of the K clusters.

After the partitioning process, each agent will be assigned to each partition. And without any influence or knowledge about another agents, the patrol task will be performed in its partition.

4 Results

4.1 Experimental Scenario

To evaluate the different alternatives, we developed a *graph generator* that generates instances of a patrolling problem. We will now describe the way graphs are generated. Parameters of the generation process will be written in uppercase, and are defined manually.

1. Randomly decide the number of vertices $n \leq MAX_n$ that the graph will contain. The vertices will occupy a 2D square of size MAX_c
2. Each vertex $v \in V$ have their 2D coordinates chosen randomly provided that different vertexes are sufficiently distant from one another, that is $\forall (v_1 \neq v_2) \in V, dist(v_1, v_2) \geq MIN_{dist}$ where $dist(v_1, v_2)$ is the euclidean distance.
3. Create edges between each pair of vertexes and randomly select $elim_v < CH_{elim}$ of those edges to be eliminated, but keeping the graph connected.
4. For each edge e that remains in E, calculate the euclidean distance between vertexes $W - e$. With probability $1 - CH_d$ the weight of the edge will be W_e and with probability CH_d the weight will be the distance multiplied by a distortion D_e randomly selected from 1 to MAX_d.

4.2 Simulation Details

In our experiments we used 500 graphs, the minimal distance between vertexes (MIN_{dist}) was 2, the size of the 2D square (MAX_c) was defined as 100×100. Each graph had between (MIN_n) 80 and (MAX_n) 100 vertexes. The chance of an endge has its weight distorted (CH_d) was 15% and the distortion (MAX_d) was at most 20%. The chance of an edge been removed (CH_{elim}) was 5%. Each experiment were run with 2, 4 and 6 agents. For each of the 500 evaluation scenarios created, we ran all strategies. Each simulation had a total time execution limited by $100 \times TSP_{cycle}$, where TSP_{cycle} is the time needed to travel the TSP-cycle of the graph. Each time we used the k-means to partition a graph, the algorithm was executed five times, with different seeds, and the best solution was chosen.

For the TSP based solutions, we used TSP solvers based on 1-tree relaxation of the branch and bound algorithm (see [12]). For the multilevel graph partitioning, we used the software METIS [13].

The PRI, PWI, and PSI are evaluated on just one randomly selected vertex of the graph. To calculate the PRI we use a Monte Carlo approach - given all visit times for the vertex, we generate 50 random attacks for each simulation, and count the number of failed attacks. The PWI and PSI are calculated deterministically - for each simulation we compute the number of failed attacks, and the number of tries. We used 5 different attack intervals based on the TSP-cycle divided by the number of agents, which we abbreviate as T/n: $\frac{1}{8} \times T/n$, $\frac{1}{4} \times T/n$, $\frac{1}{2} \times T/n$, $1 \times T/n$ and $2 \times T/n$.

4.3 Results

Tables 1, 4.3 and 3 are the main results of this work. They list for all attack intervals, the average rate with which the intrusion was averted in the five best solutions found considering separately each evaluation criterion. We calculated this average rate considering only experiments that have at least one attack try. The tables are ordered by decreasing value of general effectiveness (the average for all values). Table 1 is the data for PRI, table 4.3 for PWI, and table 3 for PSI.

Table 1. PRI

Number of Agents	Partition Scheme	Sequencing Algorithm	PRI(%)				
			$\frac{1}{8} \times \frac{T}{n}$	$\frac{1}{4} \times \frac{T}{n}$	$\frac{1}{2} \times \frac{T}{n}$	$1 \times \frac{T}{n}$	$2 \times \frac{T}{n}$
2	K-Means	Original TSP	12	24	47	91	100
	K-Means	TSP local visits	11	22	44	84	99
	Equal distance cycle	Original TSP	12	23	43	76	99
	K-Means	TSP rank	11	22	43	79	98
	Random cycle	Original TSP	12	23	43	74	99
4	K-Means	Original TSP	12	23	47	87	100
	K-Means	TSP local visits	11	22	44	83	99
	K-Means	TSP rank	11	22	42	77	98
	Hierarchical	Original TSP	12	24	44	68	93
	Equal distance cycle	Original TSP	12	23	41	69	94
6	K-Means	Original TSP	13	25	49	87	100
	K-Means	TSP local visits	12	24	48	85	100
	K-Means	TSP rank	12	23	45	78	98
	Hierarchical	Original TSP	12	24	46	76	96
	Hierarchical	TSP local visits	12	23	45	73	94

Table 2. PWI

Number of Agents	**Partition Scheme**	**Sequencing Algorithm**	**PWI(%)**				
			$\frac{1}{8} \times \frac{T}{n}$	$\frac{1}{4} \times \frac{T}{n}$	$\frac{1}{2} \times \frac{T}{n}$	$1 \times \frac{T}{n}$	$2 \times \frac{T}{n}$
2	Random cycle	Original TSP	7	12	25	50	100
	Equal distance cycle	Original TSP	6	11	25	50	100
	Equal distance cycle	TSP rank	7	14	26	50	89
	Random cycle	TSP rank	7	13	26	50	89
	Random cycle	TSP local visits	6	12	24	46	89
4	Random cycle	Original TSP	8	17	32	57	88
	Equal distance cycle	Original TSP	8	16	32	57	88
	Random cycle	TSP rank	9	17	32	56	85
	Equal distance cycle	TSP rank	9	17	32	56	85
	Equal distance cycle	TSP local visits	9	16	31	54	83
6	Equal distance cycle	Original TSP	9	19	35	60	87
	Random cycle	Original TSP	9	19	35	59	87
	Random cycle	TSP rank	10	19	34	58	85
	Equal distance cycle	TSP rank	10	19	34	58	84
	Equal distance cycle	TSP local visits	9	18	33	56	83

Table 3. PSI

Number of Agents	**Partition Scheme**	**Sequencing Algorithm**	**PSI(%)**				
			$\frac{1}{8} \times \frac{T}{n}$	$\frac{1}{4} \times \frac{T}{n}$	$\frac{1}{2} \times \frac{T}{n}$	$1 \times \frac{T}{n}$	$2 \times \frac{T}{n}$
2	Random cycle	TSP rank	6	10	17	28	81
	Equal distance cycle	TSP rank	6	11	18	28	72
	Equal distance cycle	TSP local visits	6	12	22	29	44
	Random cycle	TSP local visits	5	11	20	27	43
	Multilevel partition	TSP local visits	1	4	7	10	56
4	Random cycle	TSP rank	8	16	29	51	100
	Equal distance cycle	TSP rank	9	16	29	50	92
	Random cycle	TSP local visits	8	15	26	50	88
	Equal distance cycle	TSP local visits	8	14	26	47	80
	Equal distance cycle	TSP local changes	7	13	24	44	85
6	Random cycle	TSP rank	10	18	33	59	100
	Equal distance cycle	TSP rank	10	18	33	57	100
	Random cycle	TSP local visits	9	16	31	55	100
	Equal distance cycle	TSP local changes	8	15	28	48	94
	Random cycle	TSP local changes	8	15	27	48	90

5 Discussion and Conclusions

The reliable patrolling is a complex problem, requiring solutions that integrate efficiency and unpredictability. The main contribution of this paper consists into presenting 3 new metrics to evaluate the patrolling problem. Each metric considers different kinds of invaders. Based on this metrics we proposed and compared various partition schemes and sequencing algorithms.

Our results point out that for invaders that act randomly - the attack is made without any knowledge about the agents - the traditional partitioning schemes are more effective than use non-partitioning strategies. However, when the attacker have some information about the agents, like an historical of visits to a specific place, or even if the invader can only perceives when an agent visit some place, the non-partitioning strategies perform better. It is also important to notice that in the non-partitioning strategies the best equal distance distribution of the agents not necessarily guarantee a better solution, since a random distribution can also contribute to the unpredictability of the strategy. Another important result is that for invaders that use statistical information to plan his actions the sequencing algorithm that achieve the best results were the TSP rank, which is not a static solution, like the original TSP strategy. This corroborate our assumption that unpredictability is an essential characteristic to the patrolling task. And as a general rule, random walk based sequencing algorithms, although very unpredictable, have so long periods between visits, that they are not usefull to avert any of the three attackers modeled.

When the attacker acts randomly or with very restricted information, perform the TSP cycle over and over was the best solution found. This happens because this approach finds the cycle that covers all the nodes with the minimal time needed, so it maximizes the number of times that all the vertexes will be visited, raising the chances that an agent averts an invasion from a random or almost-random invader.

The solutions proposed in this paper are mainly centralized solutions. For example, the TSP with local visits selection depends on calculating the TSP solution to the graph, which requires not only global knowledge of the graph but enough computational power. But once the solution is computed and distributed to the agents, they each perform based on local decisions, regardless of the other agent's decisions. Other architectures, for example, the global random architecture are also based on local decisions, and require global knowledge of the graph, but are not that computationally expensive.

From one point of view, if the terrain being patrolled is static, and if the patrolling will last for a long time, then it is probably worth to compute the "best" solution to the problem in a centralized way. In situations in which a global knowledge is not possible, or it is not worth to gather all the local knowledge into a global one, and spend time computing the best solution, a distributed way of achieving the solution could be interesting, and our results can be seen as an upper bound to what can be achieved with the distributed problem solving.

Extensions to this work include, for instance, analysis of more other strategies, modifications on the scenario generator to include new specific cases; inclusion of new characteristics to the patrolling task, for example, prioritized regions and dynamic topologies. Another important extension is to study the scenario where there is redundancy in the number of agents that visits a place, so that the compromise of one/few patrolling agent does not compromise the system as a whole.

References

1. Hespanha, J., Kim, H.J., Sastry, S.: Multiple-agent probabilistic pursuit-evasion games. In: Proceedings of the 38th IEEE Conference on Decision and Control, 1999, vol. 3, pp. 2432–2437 (1999)
2. Flint, M., Polycarpou, M., Fernandez-Gaucherand, E.: Cooperative control for multiple autonomous uav's searching for targets. In: Proceedings of the 41st IEEE Conference on Decision and Control, 2002, vol. 3, pp. 2823–2828 (2002)
3. Subramanian, S., Cruz, J.: Adaptive models of pop-up threats for multi-agent persistent area denial. In: Proceedings. 42nd IEEE Conference on Decision and Control, 2003, December 9-12, vol. 1, pp. 510–515 (2003)
4. Grace, J., Baillieul, J.: Stochastic strategies for autonomous robotic surveillance. In: 44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC 2005, pp. 2200–2205 (2005)
5. Machado, A., Ramalho, G., Zucker, J.D., Drogoul, A.: Multi-agent patrolling: An empirical analysis of alternative architectures. In: Sichman, J.S., Bousquet, F., Davidsson, P. (eds.) MABS 2002. LNCS (LNAI), vol. 2581, pp. 155–170. Springer, Heidelberg (2003)
6. Chevaleyre, Y., Sempe, F., Ramalho, G.: A theoretical analysis of multi-agent patrolling strategies. In: AAMAS 2004: Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems, vol. 3, pp. 1524–1525. IEEE Computer Society, Los Alamitos (2004)
7. Almeida, A., Ramalho, G., Santana, H., Tedesco, P.A., Menezes, T., Corruble, V., Chevaleyre, Y.: Recent advances on multi-agent patrolling. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 474–483. Springer, Heidelberg (2004)
8. Santana, H., Ramalho, G., Corruble, V., Ratitch, B.: Multi-agent patrolling with reinforcement learning. In: AAMAS 2004: Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems, vol. 3, pp. 1122–1129. IEEE Computer Society, Los Alamitos (2004)
9. Menezes, T., Tedesco, P., Ramalho, G.: Negotiator agents for the patrolling task. In: Sichman, J.S., Coelho, H., Rezende, S.O. (eds.) IBERAMIA 2006 and SBIA 2006. LNCS (LNAI), vol. 4140, pp. 48–57. Springer, Heidelberg (2006)
10. Barnes, G., Feige, U.: Short random walks on graphs. SIAM Journal on Discrete Mathematics 9(1), 19–28 (1996)
11. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Computing Surveys 31(3), 264–323 (1999)
12. Volgenant, A., Jonker, R.: A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. European Journal of Operational Research 9, 83–89 (1982)
13. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20(1), 359–392 (1998)