Parallel Multi-speed Pursuit-Evasion Game Algorithms

Renato Fernando dos Santos[1], **Marcos Augusto Menezes Vieira**[2]

[1]Campus Coxim – Instituto Federal de Mato Grosso do Sul (IFMS)
Rua Salime Tanure, S/N, Bairro Santa Tereza – Coxim – MS – Brazil

[2]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Av. Antonio Carlos, 6627 - ICEx Pampulha – Belo Horizonte – MG – Brazil

renato.santos@ifms.edu.br, mmvieira@dcc.ufmg.br

PhD Student* - Thesis Defended on 06/02/2023.

*This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract. *Pursuit-Evasion Game (PEG) is important in the robotics field. Determining the optimal strategy, such as the minimum number of pursuers needed, often require exponential time. We propose a novel parallel algorithm that significantly reduces the computation time for PEG, making it feasible to analyze games with a large number of states and transitions. This algorithm also extends to the heterogeneous/multi-speed players and incorporates advanced strategies. Additionally, we introduce a resource allocation algorithm for heterogeneous multi-agent teams, ensuring efficient resource sharing and real-time agent replacement to maintain fault tolerance. Our simulations demonstrate that our parallel algorithm significantly outperforms existing approaches, achieving up to 8.13 times speedup compared to the state-of-the-art. Furthermore, our algorithms enhance the scalability and practical applicability of solving PEGs.*
Key-words: *Pursuit-Evasion Game, Parallel Algorithm, Multi-agent Systems, Heterogeneous Robots, Multi-Team Resource Allocation.*

Resumo. *O Jogo de Perseguição-Fuga (PEG) é importante no campo da robótica. Determinar a estratégia ótima, como o número mínimo de perseguidores necessário, frequentemente exigindo tempo exponencial. Propomos um novo algoritmo paralelo que reduz significativamente o tempo de computação para PEG, tornando possível analisar jogos com um grande número de estados e transições. O algoritmo também se estende aos jogadores heterogêneos/multi-velocidade e incorpora estratégias avançadas Além disso, introduzimos um algoritmo de alocação de recursos para equipes multi-agentes heterogêneas, garantindo o compartilhamento eficiente de recursos e a substituição em tempo real de agentes para manter a tolerância a falhas. Nossas simulações demonstram que nosso algoritmo paralelo supera significativamente as abordagens existentes, alcançando até 8,13 vezes mais velocidade em comparação com o estado da arte. Além disso, nossos algoritmos melhoram a escalabilidade e a aplicabilidade prática na resolução de PEGs.*
Palavras-chave: *Jogo de Perseguição e Fuga, Algoritmo Paralelo, Sistemas Multi-agentes, Robôs Heterogêneos, Alocação de Recursos para Multi-Times.*

1. Introduction

Pursuit-evasion games (PEGs) and resource allocation for heterogeneous multi-agent systems are two prominent areas of research that have garnered significant attention in recent years. PEGs involve the interaction between pursuers and evaders in a dynamic environment, where the pursuers aim to capture or intercept the evaders while the evaders strive to avoid capture. PEGs have found applications in diverse domains such as surveillance, search and rescue operations, and network deployment, where effective strategies for pursuing and evading entities are crucial.

Heterogeneous multi-agent systems, involve the coordination and collaboration of agents with diverse capabilities, resources, and objectives to accomplish complex tasks. The allocation of resources among these agents plays a vital role in optimizing task performance and ensuring fault tolerance. Resource allocation for heterogeneous multi-agent systems requires efficient mechanisms to distribute resources effectively, maintain task resolution in the presence of failures, and adapt to dynamic environments.

The effective resolution of pursuit-evasion games and resource allocation in heterogeneous multi-agent systems pose several challenges that demand innovative approaches and algorithms. Traditional methods for solving pursuit-evasion games often suffer from scalability limitations, hindering their applicability to large-scale and complex scenarios. Similarly, resource allocation in heterogeneous multi-agent systems requires mechanisms that can handle the dynamic nature of tasks and agents, ensuring efficient utilization of resources and fault tolerance. The motivation behind the thesis was to address these challenges and contribute to the fields of pursuit-evasion games and resource allocation for heterogeneous multiagent systems. By developing scalable algorithms for solving pursuit-evasion games and efficient resource allocation techniques for heterogeneous multi-agent teams, we aim to enhance task performance, optimize resource utilization, and improve the fault tolerance of multi-agent systems.

The primary objective of the thesis is to propose and validate algorithms and techniques that enhance pursuit-evasion game strategies and resource allocation in heterogeneous multi-agent systems. This includes developing a parallel algorithm for solving pursuit-evasion games to handle large-scale scenarios and improve computational efficiency. Additionally, we aim to extend the pursuit-evasion game framework to incorporate multi-speed players and strategies like the pac-dot strategy, which enhances evader lifetime and optimal strategy computation. Another objective is to design a resource allocation algorithm for heterogeneous multi-agent teams, enabling efficient resource sharing and real-time agent replacement to improve task performance and fault tolerance. Finally, we evaluated and validate these proposed algorithms and techniques through extensive simulations, demonstrating their effectiveness and scalability.

The thesis makes significant contributions to the fields of discrete pursuit-evasion games and resource allocation for heterogeneous multi-agent teams. The key contributions of this research are as follows: We propose a novel parallel algorithm that improves the scalability and computational efficiency of solving pursuit-evasion games on graph-based environments. The algorithm enables the analysis of games with a large number of states and transitions, providing more comprehensive insights into pursuit and evasion strategies. We extend the pursuit-evasion game framework to support multi-speed players and incorporate the pac-dot strategy. These extensions enhance the evader lifetime

Table 1. Categorization of PEG Related Work.

Criterion	[1]	[2]	[3]	[4]	[5]	[6]	our work
Pursuer to evader ratio	> 1	$= 1$	> 1	> 1	$>= 1$	$= 1$	$>= 1$
Players Configuration	MPME	SPSE	MPSE	MPSE	MPME	SPSE	MPME
Environment	Graph	Graph	Polygon	Polygon	Polygon	Graph	Graph
Environment Type	Discrete	Discrete	Discrete	Continuous	Continuous	Discrete	Discrete
Pursuer Visibility	Full	Local	Full	Full	Full	Local	Full
Evader Visibility	Full	Local	Full	Full	Full	Full	Full
Multi-Speed	Yes	Yes	Not	Not	Yes	Not	Yes
Capture Guaranteed	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Optimal	Yes	Yes	Not	Not	Yes	Yes	Yes
Parallel Algorithm	Not	Not	Not	Not	Not	Not	Yes

and enable the computation of optimal strategies that consider different player speeds and objectives. We develop a resource allocation algorithm that enables the construction of teams composed of heterogeneous agents and ensures efficient resource sharing. The algorithm incorporates real-time agent replacement to address failures, facilitating seamless task resolution and fault tolerance. Extensive simulations and evaluations are conducted to validate the proposed algorithms and techniques. The performance and effectiveness of the parallel algorithm for pursuit-evasion games and the resource allocation algorithm for heterogeneous multi-agent teams are compared with existing approaches, demonstrating their scalability, efficiency, and practical applicability.

The thesis has led to two significant publications: one in a specialized robotics journal[1] and one presented at a conference[2], both rated 'A1' by Qualis Capes, with the journal having a JCR Impact Factor of 4.3.

2. Literature Review

We begin this section by presenting the related work of the Pursuit-Evasion Game Framework. To situate our work in the literature, we used nine criteria to classify the type of PEG (1): the pursuer-to-evader ratio; players configuration indicates one or more pursuers and one or more evaders; environment: graph or polygon; type: discrete or continuous; vision: full or partial; multi-speed: players move at different speeds; capture guarantee and optimal strategy. The related work are presented in Table 1 as: [1]–[Vieira et al. 2009]; [2]–[Krishnamoorthy et al. 2013]; [3]–[Huang et al. 2011]; [4]–[Yan and Jiang 2017]; [5]–[Makkapati and Tsiotras 2020]; [6]–[Zhang et al. 2019].

Pursuit-Evasion Game (PEG) is a well-studied topic in the robotics literature [Chung et al. 2011, Bopardikar et al. 2008, Pan et al. 2012, Vieira et al. 2009]. The huge interest for PEGs in robotics stems from its application to multi-robot problems such as surveillance, search and rescue, boundary defense, and network deployment [Tekdas et al. 2010]. In addition, PEGs are used to obtain results on the worst-case performance of robotic systems [Chung et al. 2011]. Common approaches for solving and analyzing PEGs are based on game theory, and these approaches can be traced back to the seminal work of Isaacs on differential games [Isaacs 1999]. Various versions of PEGs were introduced in the robotics literature, including continuous-

[1]dos Santos, R. F., Ramachandran, R. K., Vieira, M. A., and Sukhatme, G. S. (2023). Parallel multi-speed pursuit-evasion game algorithms. Robotics and Autonomous Systems, 163:104382.

[2]dos Santos, R. F., Ramachandran, R. K., Vieira, M. A. M., and Sukhatme, G. S. (2020). Pac-man is overkill. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 11652–11657.

time PEGs [Yamaguchi 1999], discrete-time PEGs [Bopardikar et al. 2008], and discrete PEGs [Parsons 1978]. In the thesis, we focused on discrete PEGs and we are mainly based on the formulations presented in [Vieira et al. 2009]. There are many approaches to solving PEGs, each designed using a different method, as can be seen in the thesis.

In [Vieira et al. 2009] was proposed the design and implementation of an exact and optimal algorithm to compute discrete pursuit-evasion games, but it does not scale beyond a small number of robots. This work was the main basis of the Parallel Algorithm and its extensions for calculating PEG.

Table 2. Categorization of Related Work.

Criteria	[1]	[2]	[3]	[4]	[5]	Our work
Types of Agents	Heterogeneous	Homogeneous	Heterogeneous	Heterogeneous	Heterogeneous	Heterogeneous
Fault Tolerant	Not	Not	Yes	Yes	Yes	Yes
Static/Dynamic	Static	Dynamic	Dynamic	Static	Dynamic	Dynamic
Multi-team	Not	Yes	Not	Not	Not	Yes

Table 2 we categorized by characteristics, the related work about Resource Allocation for Heterogeneous Multi-agents Teams Framework. In line 1 the types of agents can be heterogeneous (Het.) or homogeneous (Hom.). Next, if the work is fault tolerant or not. In the third line, whether the allocation of resources is static (prior to the start of the task execution) or dynamic (before and during the task execution). Next, if the approach is for multi-teams of agents (yes) or if it is for only one team (no). Our work is for heterogeneous, fault-tolerant agents, with dynamic resource allocation, and multi-teams. The related work in the Table 2 are: [1]–[Abbas and Egerstedt 2014]; [2]–[Makkapati and Tsiotras 2019]; [3]–[Notomista et al. 2019]; [4]–[Varadharajan et al. 2019]; [5]–[Ramachandran et al. 2020]

In [Abbas and Egerstedt 2014] the work presents a framework that analyzes the assignment of multiple resources to nodes and its effect on the overall network heterogeneity, specifically focusing on resource allocation among nodes and their neighborhoods within a single network. This work serves as the foundation for the Resource Allocation framework. The difference between our proposal and Abbas and Egerstedt's proposal is that we focus on working with several teams simultaneously, instead of just one, expanding cooperation to support fault tolerance and enable the exchange of resources among teams, both before and during the execution of the PEG.

3. Methodology

We designed a parallel algorithm and extended it for multi-speed agents, the pac-dot strategy, and the prune pac-dot strategy. Additionally, we designed an algorithm for resource allocation among teams of multi-agents. The resource allocation algorithm was applied to maximize the number of teams as completely cooperative networks, where all agent/node have all the resources available in your neighborhood. The algorithm was applied to be fault tolerant for three cases (see Section 7). We developed a discrete simulator to evaluate PEGs and resource allocation in different topologies. The algorithm and simulator were encoded in the Python 3 language.In addition to the native resources of the Python language, we used the Joblib package. More specifically, we used the class *joblib.Parallel* to implement parallel functions. We used three computers to simulate the algorithms:

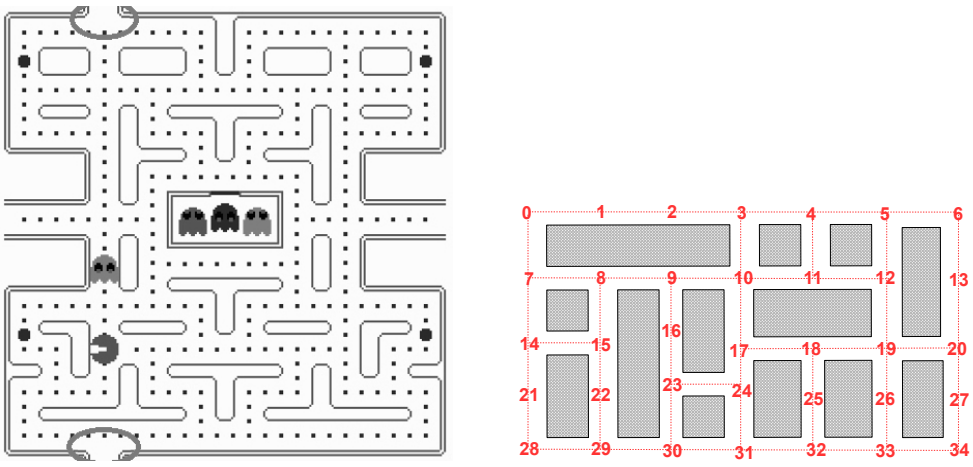
1. Processor Intel XEON E5-2630 v3 2.4GHz 32 cores, 128GB RAM and 12TB HD;
2. Intel Core i7-6800K 3.4GHz 12 cores, 64GB RAM, 1TB HD.
3. Processor AMD EPYC 7282 2.80 GHz 32 cores, 128 GB RAM, and 240 GB SSD.

All simulations of Table 3 and Table 4 were performed on computer 1. Pac-man simulations for multi-speed pursuers were performed on computer 2. All simulations of the Table 5, speedup evaluations (Figure 5) and Table 8 (Section 7) were simulated on computer 3. Our database of topological maps is mostly public, composed as follows: some were extracted from Pac-Man games, others are floor plans of real places, and others were created by us, alluding to real places.

4. Parallel Algorithm

The parallel algorithm was designed to minimize the time of capture in a PEG with pursuers and an evader playing optimal strategies [Vieira et al. 2009]. The inputs are a set of agents (pursuers and an evader) and a topological map. From the inputs, all states are generated. A state is defined as the players' positions on the map at a given time. Capture states are those where at least one pursuer and the evader occupy the same node, with a cost of zero. All possible state combinations are generated, followed by transitions, which are viable moves from one state to other. The solution from all states to a capture state is computed using a minimax tree [Russell and Norvig 2009]. The algorithm outputs all optimal solutions (trajectories) from any starting positions until the evader is captured.

5. PEG Framework: Simulation, Evaluation, and Results



(a) Pac-Man Toroidal **(b)** Building 2.

Figure 1. Two of the Evaluated Topologies.

In this section, we present the most relevant simulations, evaluations and results. First, we discuss the speedup evaluation of the parallel algorithm. The remaining subsections present PEG computation results for the parallel algorithm and its extensions: the multi-speed algorithm, the pac-dot algorithm, and the pac-dot algorithm with pruning. We evaluate many topologies, but we focus on the main results in this work, for the topologies presented in Figure 1: Pac-Man game (Figure 1a) and Building 2 (Figure 1b).

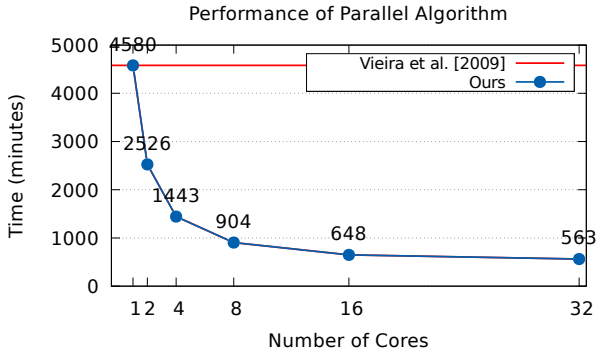


Figure 2. Serial execution ([Vieira et al. 2009]) performance in one core vs. parallel algorithm execution (Ours) for 2, 4, 8, 16 and 32 cores.

5.1. Parallel Algorithm Speedup

We present the performance of the parallel algorithm (section 3.3 of the thesis). In Figure 2, we have the time (in minutes) of serial execution for 1 core (the execution time in [Vieira et al. 2009]) and the time of parallel executions for 2 cores up to 32 cores (the time of our executions). Executions were performed on the Pac-Man topology (Figure 1a) with 2 pursuers and an evader. All players with speed = 1. In Figure 2 we can see that as the number of cores doubles, the curve flattens (blue line points) in comparison to 1 Core line (red line). This happens with the growing number of cores, as the overhead for the creation/maintenance of the pool of threads/processes also grows.

We used Amdahl's law to calculate speedup, which measures the performance improvement of a system with enhancements. Speedup is computed as $S = \frac{T(1)}{T(2)}$, where $T(1)$ is the time without improvement and $T(2)$ is the time with improvement. A speedup of 1 indicates no improvement, while a speedup greater than 1 shows how much faster the improved version is. For our algorithm, the speedup for 2 cores was 1.90 (parallel) and 1.81 (total), and for 32 cores, it was 15.10 (parallel) and 8.13 (total).

5.2. Parallel Algorithm

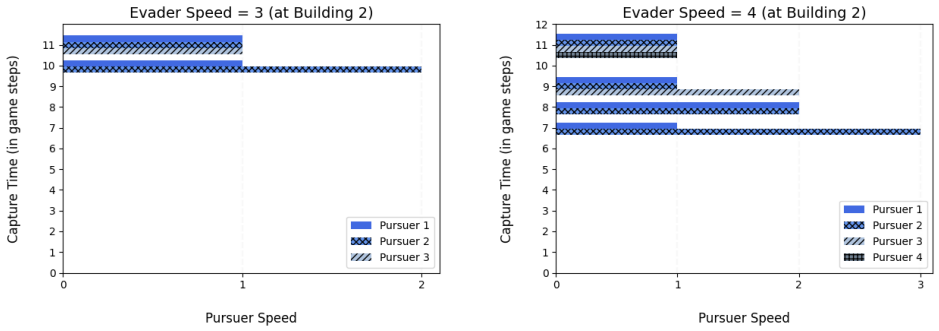
Table 3. Topologies and simulation results - Parallel Algorithm without pac-dot.

Topology	Players	Vertices	States	Transitions	Time to generate Game Graph (min)	Cost Calculation Time (min)	Total Execution Time (min)	Cost/Steps	Diameter
Pac-Man	3	290	24389000	313285590	16	272	288	45	51
Pac-Man	3	290	24389000	313285590	16	272	288	65	51
Pac-Man Toroidal	3	290	24389000	402542566	23	355	378	73	38
Pac-Man Google	3	317	31855013	402542566	29	431	460	64	49
Ms. Pac-Man	3	308	29218192	376875135	39	831	870	69	44

Table 3 displays the PEG simulation results for the described topologies. Columns are defined as follows: Column 1 identifies the topology, Column 2 shows the number of players (pursuers and evader(s)), Column 3 lists the number of vertices, Columns 4 and 5 indicate the number of states and transitions needed for the game graph, Column 6 shows the time to generate the game graph, Column 7 shows the time to calculate the cost, Column 8 is the total execution time (sum of Columns 6 and 7), Column 9 presents the worst-case steps (cost) to capture the evader, and Column 10 shows the topology graph's diameter. The cost reflects the evader's attempts to delay capture, while the diameter represents the longest path in the graph.

Pac-man topology considering the original starting positions of the game (first line of Table 3), requires 45 steps to capture the evader and only needs 2 pursuers. Considering the diameter distance between pursuers and the evader as the starting position (second line of Table 3) requires 65 steps to capture, it also only needs 2 pursuers to capture the evader. These evaluations show that Pac-Man is overkill with 4 ghosts (pursuers) since it only needs 2 pursuers to capture an evader.

5.3. Heterogeneous / Multi-speed Player



(a) Evader speed = 3 **(b)** Evader speed = 4

Figure 3. Evaluation of Building 2 topology for multi-speed players.

We evaluated the topologies Building 1 and Building 2 for heterogeneous players, but we presented here only (Figure 1b). We simulated each topology for evader speed equal to 3 and 4. For all simulations, pursuers, and the evader started at the opposite end of the topology diameter. Pursuers follow two criteria: the minimum number of pursuers to guarantee the evader capture and the sum of the speed of all pursuers is equal to the evader speed. Figures 3a-3b show the results of the simulations mentioned above. Each bar represents a pursuer and a grouping of bars represents a team of pursuers. The label on X-axis is the cost to the pursuers capturing the evader and the label on Y-axis is the pursuers speed. In the Figure 3b the evader speed is 4, with two pursuers with 1 and 3 speeds respectively the cost to capture is 7 game steps and the cost is 11 to four pursuers with speed 1 each. We can observe in the simulations (Figures 3a–3b) that a configuration where a small number of pursuers in which one pursuer has speed close to the evader speed performs better than configurations with more than two pursuers with greater equality in the speed distribution.

5.4. PEG with Pac-dot

Table 4. Topologies and simulation results - Parallel Algorithm with pac-dot.

Topology	Players	Vertices	States	Transitions	Number of Pac-dot	Immunity Time	Pac-dot Position	Cost/Steps	Diameter
Reduced Pac-Man	3	102	12734496	166569444	1	10	square	34	21
Reduced Pac-Man	3	102	36081072	472402512	2	10	triangle/square	39	21
Reduced Toroidal Pac-Man	3	102	12734496	168394616	1	10	square	36	21
Reduced Toroidal Pac-Man	3	102	36081072	477578800	2	10	circle/square	50	21

Table 4 shows the results of the PEG with pac-dot simulation for Reduced Pac-Man, Reduced Toroidal Pac-Man topologies (page 52 of the thesis). This topology was simulated on computer 1 and computer 2. Columns are defined from left to right.

Columns 1-5 have the same meaning as in Table 3. Column 6 shows the number of pac-dot of the simulation. Column 7 presents the evader immunity time duration. Column 8 depicts the position of each pac-dot in a simulation, differentiated by geometric shapes. Columns 9 and 10 have the same meaning as in Table 3. Reduced Pac-Man topology with 1 and 2 pac-dots, result in a cost of 34 and 39 respectively. Reduced Toroidal Pac-Man topology with 1 and 2 pac-dots, result in a cost of 36 and 50 respectively. Note in Table 3 that the cost is 21 for these two topologies without pac-dot. Although the immunity time is 10 time steps for both topologies, the resulting cost is not $21+10$ and $21+20$. This is due to the pursuers escaping the evader during the immunity time. Despite the immunity time, all players performed an optimal strategy.

5.5. Pruning

Table 5. Simulation results - Parallel Algorithm with pac-dot and pruning.

Topology		States			Transitions			Simulation Time		
	Pac-dot Number	Without Pruning	Pruning	Ratio (P/WP)	Without Pruning	Pruning	Ratio (P/WP)	Without Pruning	Pruning	Ratio (P/WP)
Reduced Pac-Man	1	12734496	3891096	30.56%	166569444	50159972	30.11%	03:02:55	00:45:56	25.11%
Reduced Pac-Man	2	36081072	10351980	28.69%	472402512	133623512	28.29%	09:04:36	02:31:37	27.84%
Reduced Toroidal Pac-Man	1	12734496	3891096	30.56%	168394616	59636060	35.41%	02:47:18	00:48:33	29.02%
Reduced Toroidal Pac-Man	2	36081072	12744900	35.32%	477578800	166293280	34.82%	07:21:01	03:27:07	46.96%
Reduced Toroidal Pac-Man	4	176160528	68125392	38.67%	2332709344	891087400	38.20%	43:59:28	17:20:52	39.43%
Pac-Man	1	292668000	60720238	20.75%	3781151956	777796844	20.57%	-	45:14:50	-
Pac-Man	2	829226000	145324876	17.53%	10716891864	1858045016	17.34%	-	129:36:55	-

Table 5 shows the results of the PEG with pac-dot in comparison to the PEG with pac-dot with pruning simulation. In the first line, the Table is divided in four groups: Topology; States; Transitions; and, Simulation Time. Topology has two columns, on the left is the topology name and at right the number of pac-dots configured. States and transitions took the number of states/transitions (respectively) generated by pac-dot without pruning simulation (PW) and pac-dot with pruning simulation (P), hence ahead, namely pruning. The third column of States and Transitions part took the ratio between P/PW. In the Simulation Time the three columns are the time to compute the game without pruning, with pruning, and the ratio of the pruning. Cells with - (hyphen) mean that it was not possible to complete the execution due to the lack of computational resources, such as primary or secondary memory space. The Pac-Man topology simulated with 2 pac-dots had a ratio of 17.53% in the number of states, a gain of 82.47%. The gain was similar to the number of transitions, which in absolute numbers exceeds 10.5 billion of transitions.

6. Resource Allocation Algorithm

Let G be the set of heterogeneous networks, each indexed by $i = \{1,...,g\}$. Each G_i has an adjacency matrix A and a resource matrix C. From these matrices, the resource distribution matrix ϕ and the deficiency color matrix U are generated. We assume r (the number of different resources) is identical for all networks.

Networks are classified as sufficient, provider, or receiver. A sufficient network has all r resource and the minimum number of agents $c(T)$ required for capture in the PEG for topology T. A provider network has extra agents with spare resources and more than $c(T)$ agent, allowing it to transfer agent to a compatible receiver network. A receiver network lacks resources to be a CHCN as some like resource or to reach $c(T)$. Then, agent transfers between networks are made, necessary edges are added to achieve CHCN status, redundant edges are removed, and the networks are prepared to run the PEG.

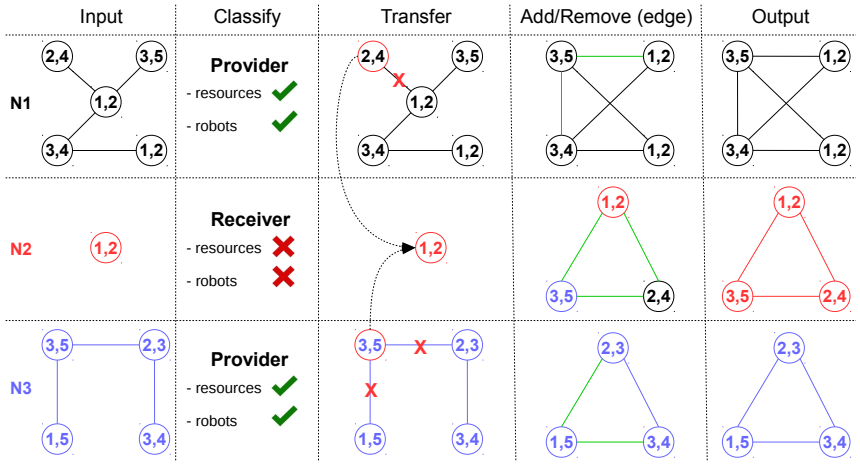


Figure 4. Resource (re)distribution algorithm Steps.

7. Resource Allocation Framework: Simulation, Evaluation and Results

In this section, we present the main evaluation and results carried out for some sets of heterogeneous agent networks to evaluate the distribution of resources to make them completely heterogeneous cooperative networks (CHCNs). Then, these networks play PEG to evaluate the fault tolerance approach. The presented simulations are carried out for the topology Building 2 (Figure 1). Consider the terms network and team as interchangeable.

Table 6. List of resources/sensors.

Id	Resource/Sensor	Description
1	Wireless/4G/5G/LoRa	Communication
2	GPS/Beacon	Localization
3	Odometry/Accelerometer	Self-localization
4	Ultrasonic	Distance Measurement
5	Laser	Distance and sensing

For simulations with heterogeneous agents and their capabilities, the Table 6 shows the resources and sensors embedded in agents. All agents move at the same speed, one hop at a time. Agents with communication resources inform other networks about the current status and identify broken agents. Resource two uses GPS for outdoor or beacons for indoor environments to report the agent's location to the central controller. Agents without resource two rely on nearby agents for location. Resources three, four, or five include an odometry sensor and an accelerometer for measuring displacement. Resource four helps avoid collisions, while resource five avoids collisions and measures distance.

Figure 4 illustrates the stages of the resource allocation algorithm in execution. The algorithm inputs are networks $N1$, $N2$, and $N3$, modeled as graphs. The circles denote agents, the edges represent links between agents, and the circle labels indicate the number of resources embedded in the agents. For this simulation, each network needs all the resources in Table 6 and at least two nodes, the minimum number for the target topology (Figure 1b) outputs the CHCNs to play PEG. In the PEG context, all the agents in the networks are pursuers. Initially, the algorithm performs the classification stage, where $N1$ and $N3$ have spare resources and more agents than the minimum, thus being classified as Provider networks. $N2$ is classified as a Receiver network since it lacks all

Table 7. Topology characteristics.

Topology	Nodes	Players	States (S)	Transitions (T)	T/S Ratio
Building 2	35	3	85750	798836	9.3
		4	3001250	91633892	30.53
		5	105043750	11821630076	112.5

Table 8. Fault Tolerance Evaluation for Building 2 Topology.

		Cost				
Network Id	Starting Positions	Without Failure	With Failure	Failure Step	Failure Node Label	Received Node
N1	5 6 5 9 24*	6	$8^1, 13^2, 14^3$	$5^1, 5^2, 6^3$	$(1,2)^1, (3,5)^2, (3,5)^3$	$-^1, N2(3,5)^2, N3(1,5)^3$
N2	27 25 16 21*	7	$12^2, 13^3$	$5^2, 7^3$	$(1,2)^2, (2,4)^3$	$N1(3,4)^2, N1(3,4)^3$
N3	16 16 22 32*	8	$11^2, 13^3$	$5^2, 8^3$	$(3,4)^2, (3,4)^3$	$N1(3,4)^2, N1(3,4)^3$

the necessary resources and the minimum number of agents. The next stage is to analyze and transfer the corresponding nodes from a provider to a receiver. Subsequently, $N1$ transfers the node labeled $2,4$, and $N3$ transfers the node labeled $3,5$, both received by $N2$. Marked edges with a red x represent their removal. At the add/remove stage, the added edges are green, and the black edges remain from the previous stages. No edges needed to be removed at this stage. Note that $N2$ received nodes, preserving the color of their source networks. The outputs are the CHCNs.

For PEG simulations, an evader is associated with each output network. We need to compute the PEG optimal strategy for the target topology with 3, 4, and 5 agents (or players), where 3 is the minimum number of agents (pursuers and an evader), and 5 (four pursuers and an evader) is the maximum number of players, as it is close to the computation limit of the PEG parallel algorithm. Although PEG state explosion is a scalability limiter, the bottleneck here is the number of transitions [Pferschy 1997], whose exponential growth curve is even greater (Table 7). The advantage is that once an instance of PEG is computed (the optimal solution from each state to the capture state is determined), it is not necessary to compute it again.

Table 7 presents the topology characteristics computed by the PEG algorithm: the name of the topology in the first column on the left; the number of nodes in the second column; the number of players for each PEG optimal strategy computation (3-5) in the third column; the number of states and transitions generated for players movement in the fourth and fifth columns; and, the ratio between the number of transitions and the number of states in the last column. For the Building 2 topology, with 5 players, the number of transitions is 112.5 times the number of states. The solutions for this topology were performed on computer 3, and it took about 20 to 25 hours to calculate the solution set.

For simulations, consider the three networks output by the resource allocation algorithm and their respective evaders to play PEG. Table 8 presents the results of various PEG simulations for networks $N1$, $N2$, and $N3$, evaluating the allocation algorithm for fault tolerance cases. Networks are identified in the column *Network Id*. The second column lists the start positions for each network's simulations, where each number represents an agent's position and the star symbol marks the evader. The column *Without Failure* shows the costs of the optimal PEG strategy. In the next four columns, superscript numbers indicate failure types: 1 - An agent broke, but the network retains all necessary

resources and the minimum number of agents; 2 - An agent broke and the network needs resources from a provider network; 3 - Same as case 2, but the network failed before starting the game. Cases 1 and 2 are *online* (occur during execution) and case 3 is *offline* (occurs before execution). The column *With Failure* shows the total simulation costs for each type of failure. The column *Failure Step* lists the game step at which the failure occurred for each type. For example, $N1$ failed at step 5 in case 2, meaning the network failed six steps from the goal. The last two columns indicate the broken agent's label and the received node from another network, with superscripts showing the failure type and linking provider and receiver networks. In case 1, $-^1$ denotes no agent was received, only a change from n to $n-1$ agents.

In all three failure cases, the new state can either benefit or damage the final cost, depending on the cost from the new state to capture. In case 1, the total cost increases by one step, reflecting the change from a 5-agent to a 4-agent solution. The total cost can either increase or decrease based on the capture cost from the new state with one fewer agent. For $N1$ (see Table 8), the cost decreased despite the failure. In cases 2 and 3, the final cost is the sum of the steps from the start to the failure, the steps waiting for resources, and the steps from the new state to capture. The new agent creates a new state, and the game restarts. The $N2$ and $N3$ networks (both in case 2) benefited as the total number of steps was reduced due to the new agent positions. If the provider network is available after capture, the cost is only one additional step.

The main advantage of our approach is its scalability, which allows increase in the number of networks and extending the model to include additional functionalities. The disadvantage is the limitation on expanding the number of agents per network.

8. Conclusion

In conclusion, the thesis has made significant contributions to the fields of pursuit-evasion games and resource allocation for heterogeneous multi-agent teams. The proposed algorithms and techniques have demonstrated their effectiveness and scalability through extensive simulations and evaluations. The parallel algorithm for PEGs has showcased improved scalability and computational efficiency, while the resource allocation algorithm has facilitated the formation of multi-teams with shared resources, ensuring efficient task performance and seamless agent replacement. The results obtained in our simulations provide promising evidence of the practical applicability and effectiveness of our approaches.

References

[Abbas and Egerstedt 2014] Abbas, W. and Egerstedt, M. (2014). Characterizing heterogeneity in cooperative networks from a resource distribution view-point. *Commun. Inf. Syst.*, 14(1):1–22.

[Bopardikar et al. 2008] Bopardikar, S. D., Bullo, F., and Hespanha, J. P. (2008). On discrete-time pursuit-evasion games with sensing limitations. *IEEE Transactions on Robotics*, 24(6):1429–1439.

[Chung et al. 2011] Chung, T. H., Hollinger, G. A., and Isler, V. (2011). Search and pursuit-evasion in mobile robotics. *Autonomous robots*, 31(4):299.

[Huang et al. 2011] Huang, H., Zhang, W., Ding, J., Stipanović, D. M., and Tomlin, C. J. (2011). Guaranteed decentralized pursuit-evasion in the plane with multiple pursuers.

In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 4835–4840.

[Isaacs 1999] Isaacs, R. (1999). *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. Dover books on mathematics. Dover Publications.

[Krishnamoorthy et al. 2013] Krishnamoorthy, K., Darbha, S., Khargonekar, P. P., Casbeer, D., Chandler, P., and Pachter, M. (2013). Optimal minimax pursuit evasion on a manhattan grid. In *2013 American Control Conference*, pages 3421–3428.

[Makkapati and Tsiotras 2019] Makkapati, V. R. and Tsiotras, P. (2019). Optimal evading strategies and task allocation in multi-player pursuit–evasion problems. *Dynamic Games and Applications*, 9(4):1168–1187.

[Makkapati and Tsiotras 2020] Makkapati, V. R. and Tsiotras, P. (2020). Apollonius allocation algorithm for heterogeneous pursuers to capture multiple evaders. *CoRR*, abs/2006.10253.

[Notomista et al. 2019] Notomista, G., Mayya, S., Hutchinson, S., and Egerstedt, M. (2019). An optimal task allocation strategy for heterogeneous multi-robot systems. In *2019 18th European Control Conference (ECC)*, pages 2071–2076.

[Pan et al. 2012] Pan, S., Huang, H., Ding, J., Zhang, W., Tomlin, C. J., et al. (2012). Pursuit, evasion and defense in the plane. In *2012 American Control Conference (ACC)*, pages 4167–4173. IEEE.

[Parsons 1978] Parsons, T. D. (1978). Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer.

[Pferschy 1997] Pferschy, U. (1997). Solution methods and computational investigations for the linear bottleneck assignment problem. *Computing*, 59(3):237–258.

[Ramachandran et al. 2020] Ramachandran, R. K., Pierpaoli, P., Egerstedt, M., and Sukhatme, G. (2020). Resilient monitoring in heterogeneous multi-robot systems through network reconfiguration. *IEEE Transactions on Robotics*. Submitted to.

[Russell and Norvig 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[Tekdas et al. 2010] Tekdas, O., Yang, W., and Isler, V. (2010). Robotic routers: Algorithms and implementation. *The International Journal of Robotics Research*, 29(1):110–126.

[Varadharajan et al. 2019] Varadharajan, V., Adams, B., and Beltrame, G. (2019). Failure-tolerant connectivity maintenance for robot swarms. *ArXiv*, abs/1905.04771.

[Vieira et al. 2009] Vieira, M. A. M., Govindan, R., and Sukhatme, G. S. (2009). Scalable and practical pursuit-evasion with networked robots. *Intelligent Service Robotics*, 2(4):247.

[Yamaguchi 1999] Yamaguchi, H. (1999). A Cooperative Hunting Behavior by Mobile-Robot Troops. *The International Journal of Robotics Research*, 18(9):931–940.

[Yan and Jiang 2017] Yan, F. and Jiang, Y. (2017). Pursuing a faster evader based on an agent team with unstable speeds. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, page 1766–1768, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Zhang et al. 2019] Zhang, Z., Lee, J., Smereka, J. M., Sung, Y., Zhou, L., and Tokekar, P. (2019). Tree search techniques for minimizing detectability and maximizing visibility. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8791–8797.