UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação

Coevolução entre duas Espécies de Formigas Artificiais Competindo por Recursos

Luca Gomes Urssi

São Carlos – SP

Coevolução entre duas Espécies de Formigas Artificiais Competindo por Recursos

Luca Gomes Urssi

Orientador: **Prof. Dr. Eduardo do Valle Simões**

Monografia final de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como requisito parcial para obtenção do título de Bacharel em Computação.
Área de Concentração: Sistemas Evolutivos, Computação Bioinspirada

USP – São Carlos
Novembro de 2024

Urssi, Luca Gomes
Coevolução entre duas Espécies de Formigas Artificiais Competindo por Recursos / Luca Gomes Urssi. - São Carlos - SP, 2024.
39 p.; 29,7 cm.

Orientador: Eduardo do Valle Simões.
Monografia (Graduação) - Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos - SP, 2024.

1. Sistemas Evolutivos. 2. Coevolução. I. Simões, Eduardo do Valle. II. Instituto de Ciências Matemáticas e de Computação (ICMC/USP). III. Título.

AGRADECIMENTOS

Agradecimentos ao Professor Eduardo do Valle Simões, pela atenção e pelas ótimas orientações neste trabalho.

Agradeço também a minha família e amigos pelo suporte e paciência ao longo dos anos.

RESUMO

URSSI L. G.. **Coevolução entre duas Espécies de Formigas Artificiais Competindo por Recursos**. 2024. 39 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Este trabalho visa a implantação de um software que simula a coevolução de formigueiros. A simulação envolve dois formigueiros (um azul e um vermelho), com o comportamento de cada formiga sendo governado por variáveis que determinam suas ações, como busca de alimento e reação aos rivais. Tais variáveis são otimizadas por um Sistema Evolutivo que visa ampliar a coleta de alimentos (forrageamento). Neste processo, os dois formigueiros competem entre si e provocam uma contínua adaptação de seus parâmetros com o passar das gerações. A interface visual da simulação foi desenvolvida utilizando a API OpenGL permitindo a representação gráfica dos feromônios liberados por formigas trabalhadoras e soldados. Essas marcas são exibidas em cores distintas para sinalizar diferentes finalidades, como caminhos, fontes de alimentos e alertas de invasores. Isso faz com que o software produzido possa ser utilizado didaticamente no ensino de computação evolutiva e inteligência de enxame. Experimentos foram executados para demonstrar a co adaptação dinâmica dos comportamentos de exploração e disputa por alimento. Os resultados demonstram que o sistema evolutivo foi capaz de produzir comportamentos superiores aos configurados manualmente, possibilitando a simulação da luta pela sobrevivência dos formigueiros implementados.

Palavras-chave: Sistemas Evolutivos, Coevolução.

ABSTRACT

URSSI L. G.. **Coevolução entre duas Espécies de Formigas Artificiais Competindo por Recursos**. 2024. 39 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

This work aims to implement software that simulates the co-evolution of ant colonies. The simulation involves two anthills (one blue and one red), with the behavior of each ant being governed by variables that determine its actions, such as searching for food and reacting to rivals. Such variables are optimized by an Evolutionary System that aims to expand food collection (foraging). In this process, the two ant colonies compete with each other and cause a continuous adaptation of their parameters over the generations. The simulation's visual interface was developed using the OpenGL API allowing the graphical representation of pheromones released by worker ants and soldiers ants. These marks are displayed in different colors to signal different purposes, such as paths, food sources and trespasser alert. This means that the software produced can be used didactically in teaching evolutionary computing and swarm intelligence. Experiments were carried out to demonstrate the dynamic co-adaptation of exploration and food-fighting behaviors. The results demonstrate that the evolutionary system was capable of producing behaviors superior to those configured manually, enabling the simulation of the struggle for survival of the implemented ant colonies.

Key-words: Evolutionary Systems, Co-evolution.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Motivação e Contextualização

Na área de pesquisa abordada por este trabalho, com ênfase na inteligência de enxames e sua otimização, destaca-se o artigo de Matheus Luis Oliveira da Silva (SILVA, 2022). O artigo de Silva descreve o desenvolvimento de um simulador de formigueiro com um forte foco na interatividade do usuário. No simulador, as formigas saem em busca de alimento e, com o auxílio de feromônios, encontram o caminho de volta, criando uma rota eficiente entre o formigueiro e a fonte de alimentos. O usuário tem a possibilidade de manipular a simulação diretamente, alterando, por exemplo, a posição da fonte de alimentos na interface para facilitar ou dificultar o acesso, o que permite uma análise mais detalhada do comportamento das formigas frente a novos desafios. Na conclusão do artigo, há uma seção que convida futuros alunos a darem continuidade ao trabalho, sugerindo algumas direções de aprimoramento. Uma dessas sugestões é a automatização do processo de ajuste das variáveis ótimas, através da aplicação de sistemas evolutivos. Isso permitiria que o simulador identificasse automaticamente os parâmetros mais eficientes para o comportamento das formigas, aumentando a precisão e a adaptabilidade do modelo em diferentes cenários.

Outro tema de interesse deste trabalho é a coevolução, que se integra naturalmente ao estudo dos formigueiros. A Coevolução de dois ou mais comportamentos é um processo complexo que tem sido executado na maioria das vezes em simulação como por exemplo o trabalho de Alice De Lorenci (LORENCI, 2020). A Coevolução, como no contexto de perseguição e fuga, ocorre quando um robô perseguidor tem seu grau de aptidão determinado pelo comportamento corrente do fugitivo, e vice-versa. A coevolução é um fenômeno central na ecologia e na biologia evolutiva, refletindo a interdependência entre espécies que interagem e competem por recursos limitados em um ecossistema.

Este trabalho irá analisar a evolução de dois formigueiros observando como as interações entre esses dois grupos influenciam a evolução de estratégias de sobrevivência. O projeto irá utilizar métodos de Sistemas Evolutivos (GABRIEL; DELBEM, 2008). Estes sistemas tipicamente trabalham em horizontes de aptidão estática. Alternativamente, a coevolução trabalha em horizontes de aptidão dinâmicos que se alteram continuamente. Assim, como descrito no livro *Encyclopedia of Insects* (RESH, 2009), a coevolução faz uso de relações entre sistemas que estão sendo evoluídos, de uma forma que dependam um do outro. Na Natureza, ao evoluir para

resolver um determinado problema, os organismos constantemente se adaptam uns aos outros e ao ambiente que os cercam, incluindo comida, parceiros sexuais, competidores e predadores. Camuflagem, por exemplo, pode ser a resposta de um predador para reagir a uma presa mais ágil.

1.2 Objetivos

Este trabalho tem como objetivo simular a coevolução de dois formigueiros, nos quais as formigas utilizam feromônios para se orientar, localizar alimentos e alertar sobre invasores, permitindo observar como o comportamento dos formigueiros evolui ao competirem por recursos. Adicionalmente, o trabalho inclui o desenvolvimento de uma interface visual para melhor compreensão da dinâmica entre os formigueiros.

Para o desenvolvimento da interface visual deste trabalho, foi utilizada a linguagem C++ em conjunto com a biblioteca OpenGL (WOO *et al.*, 1999). A simulação da coevolução, por sua vez, foi implementada exclusivamente em C++. Para garantir maior eficiência na execução, o trabalho foi otimizado por meio de paralelismo na CPU.

1.3 Organização da Monografia

Este trabalho é dividido em 5 Capítulos:

- Este capítulo de introdução contextualiza este trabalho e o objetivo dele.
- O segundo capítulo introduz brevemente o comportamento das formigas no qual este trabalho é baseado, uma introdução a sistemas evolutivos, o conceito de coevolução e as ferramentas utilizadas durante o desenvolvimento.
- No terceiro capítulo é explicada em detalhe a ferramenta de software desenvolvida neste trabalho.
- No quarto capítulo são discutidos os resultados das simulações executadas.
- O quinto capítulo contém a conclusão.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

2.1 Formigas

O comportamento das formigas simuladas neste trabalho foi fundamentado no artigo de Matheus Luis Oliveira da Silva (SILVA, 2022) e no livro *Encyclopedia of Insects* (RESH, 2009). A dinâmica das formigas segue um padrão similar ao descrito por Silva, no qual as formigas partem do ninho e se deslocam em direção a regiões não exploradas até encontrarem uma fonte de alimento ou decidirem retornar por terem se afastado excessivamente. Durante essa exploração, elas deixam um rastro de feromônio para marcar o caminho de volta ao ninho. Quando uma formiga encontra alimento, ela retorna seguindo o feromônio previamente depositado, enquanto libera um segundo tipo de feromônio que sinaliza o trajeto para a fonte de alimento. Ao retornar ao ninho com o alimento, a formiga imediatamente refaz o caminho, agora seguindo o rastro de feromônio específico para alcançar novamente a fonte de alimento, otimizando a coleta.

A leitura do livro *Encyclopedia of Insects* (RESH, 2009) permitiu confirmar o comportamento das formigas proposto por Silva e compreender a reação das formigas ao encontrar um possível invasor em seu território. O livro descreve que, ao identificar uma ameaça, a formiga libera um feromônio de alarme, convocando outras formigas para enfrentar o invasor. Esse mecanismo de defesa foi implementado neste trabalho, incluindo a simulação da emissão de feromônio de alarme para atrair reforços e combater ameaças detectadas.

Para facilitar a visualização na simulação, as formigas foram divididas em dois tipos: trabalhadoras e soldados. As formigas soldados, uma variante das trabalhadoras, possuem proporções diferentes, refletindo uma distinção que ocorre na natureza, permitindo observar de forma clara a resposta das formigas a uma ameaça detectada.

2.2 Sistemas Evolutivos

Durante o desenvolvimento deste trabalho, foram identificadas diversas variáveis cuja determinação dos melhores valores é complexa para otimizar o funcionamento da simulação. Por exemplo, a quantidade de feromônio liberada por uma formiga durante a exploração e o tempo necessário para sua dissipação são fatores críticos. Se o valor dessas variáveis fizer com que o feromônio permaneça no mapa por um período excessivo, o acúmulo excessivo poderá gerar confusão, dificultando a localização do ninho. Por outro lado, se esses valores resultarem em uma rápida dissipação do feromônio, as formigas poderão se perder ao afastarem-se do ninho, devido à ausência de pistas de orientação.

A solução para determinar os valores ideais é o uso de Sistemas Evolutivos (GABRIEL; DELBEM, 2008). Por meio desse método, um objetivo específico é estabelecido e, com a simulação sendo repetida diversas vezes, torna-se possível identificar os melhores valores para as variáveis, maximizando a eficiência da simulação e o comportamento desejado das formigas.

Sistemas Evolutivos são algoritmos inspirados nos princípios da evolução biológica que têm como objetivo encontrar soluções otimizadas para problemas complexos. A ideia central é imitar o processo de seleção natural, no qual as melhores soluções sobrevivem e se reproduzem para gerar novas soluções que, com o tempo, tendem a se aprimorar. Os principais componentes de Sistemas Evolutivos utilizados neste trabalho são: avaliação, seleção, reprodução e mutação.

A avaliação é a etapa em que cada solução, chamada de indivíduo, é analisada com base em um conjunto de critérios, geralmente definidos por uma função de aptidão (*fitness function*). Essa função quantifica o quão boa é uma solução para o problema em questão, atribuindo a cada indivíduo um valor numérico que representa sua aptidão. Esse valor indica se a solução é melhor ou pior em relação a outras e serve de base para o processo de seleção.

A seleção é o processo de escolha dos indivíduos mais aptos para reproduzir e gerar a próxima geração. Indivíduos com uma pontuação de aptidão mais alta têm maior chance de serem selecionados, pois representam soluções mais eficazes para o problema. A seleção pode ser realizada por métodos como o elitismo no qual o melhor indivíduo é automaticamente transferido para a próxima geração.

A reprodução é o processo em que os indivíduos selecionados geram novos indivíduos para a próxima geração. Esse processo é inspirado na recombinação genética em biologia e pode ser implementado por meio de um operador de *crossover*, no qual características de dois pais são combinadas para formar um filho. A ideia é que a recombinação de características vantajosas dos pais possa gerar uma nova solução ainda melhor.

A mutação é uma etapa essencial para garantir a diversidade e evitar que a população de soluções se torne muito homogênea, o que poderia levar a uma convergência prematura em uma solução subótima. A mutação introduz pequenas alterações aleatórias nas características de um indivíduo, permitindo que novas soluções sejam exploradas e que o sistema possa, eventualmente, escapar de mínimos locais.

Em um sistema evolutivo, essas etapas são repetidas através de várias gerações. Inicialmente, a população contém soluções aleatórias ou com baixa eficácia. A cada ciclo de avaliação, seleção, reprodução e mutação, a população se adapta progressivamente ao problema, gerando soluções mais refinadas. Com o tempo, o sistema evolutivo converge para soluções de maior qualidade, que atendem melhor aos critérios do problema.

2.3 Coevolução

Coevolução é um processo evolutivo em que duas ou mais espécies influenciam mutuamente suas adaptações ao longo do tempo. Cada espécie exerce uma pressão seletiva sobre a outra, levando a uma série de mudanças recíprocas que promovem adaptações especializadas.

Um exemplo clássico de coevolução é a relação entre plantas com flores e seus polinizadores, como abelhas. As flores evoluem cores, formas e aromas que atraem polinizadores específicos, enquanto as abelhas desenvolvem estruturas corporais e comportamentos que facilitam a coleta do néctar e pólen dessas flores. Esse processo beneficia ambos: as plantas aumentam suas chances de reprodução e as abelhas obtêm alimento. Se uma flor evoluir para ter uma forma mais tubular, isso poderá favorecer abelhas com probóscides (línguas) mais longas, criando um ciclo de adaptações que continua ao longo das gerações.

Neste trabalho, a coevolução será aplicada para modelar a rivalidade entre duas colônias de formigas que competem diretamente pelos mesmos recursos e podem entrar em confronto. Esse processo simula a dinâmica evolutiva entre as colônias, onde cada uma adapta suas estratégias de busca, defesa e resposta às ameaças da colônia rival. Assim, cada colônia exerce uma pressão seletiva sobre a outra, incentivando o desenvolvimento de comportamentos adaptativos, tanto para maximizar a eficiência na obtenção de recursos, quanto para melhorar suas táticas de defesa e ataque.

Capítulo 3

IMPLEMENTAÇÃO

Todo o *software* desenvolvido neste trabalho está disponivel como *software* livre sob licença GPL no repositório do *github*.[1]

No planejamento deste trabalho, foi inicialmente considerada a continuidade direta do trabalho de Silva (SILVA, 2022). No entanto, devido à falta de familiaridade com as ferramentas utilizadas em sua implementação, optou-se por desenvolver a simulação a partir do zero, com foco específico nas áreas de Sistemas Evolutivos e Coevolução.

3.1 Interface Visual

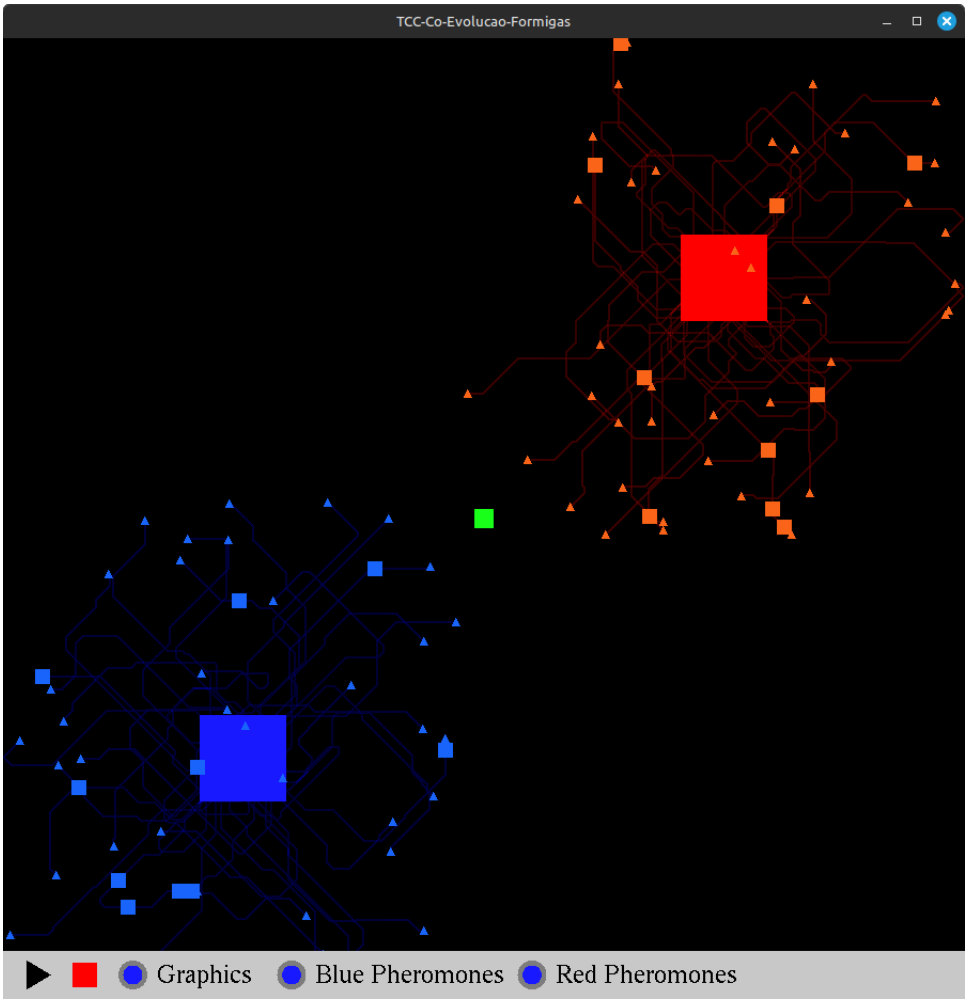
A ferramenta escolhida para implementar a interface visual do trabalho foi o OpenGL (WOO *et al.*, 1999), uma API de gráficos usada para renderizar gráficos 2D e 3D. OpenGL permite o desenho de formas básicas, linhas, e texturas com alta eficiência e controle sobre cada elemento visual. Também permite atualizar a tela para simular animações e trata das interações do usuário.

O processo de renderização de objetos na tela com a API OpenGL envolve a definição precisa das posições dos vértices da forma desejada e a especificação da cor para o preenchimento de seu volume. Por exemplo, para desenhar um quadrado, é necessário fornecer as coordenadas x e y de cada um dos quatro vértices e, em seguida, aplicar o preenchimento com a cor definida. Esse procedimento permite a criação de formas geométricas com controle direto sobre sua posição e aparência na tela.

Na Figura 1, podem ser vistos dois grandes quadrados, um vermelho e outro azul, representando os ninhos das colônias. Ao redor dos ninhos, pequenos triângulos e quadrados simbolizam, respectivamente, as formigas trabalhadoras e as formigas soldados. No centro da imagem, um quadrado verde indica a fonte de alimento que as formigas procuram durante a simulação. As linhas que compartilham a cor das formigas ao redor dos formigueiros representam os feromônios liberados no ambiente. Esses feromônios são visualizados como pequenos quadrados em cada pixel onde estão presentes, permitindo que sua distribuição e intensidade sejam observadas na simulação.

[1] <https://github.com/lucaurssi/TCC-Co-Evolucao-Formigas>

Figura 1 – Simulador com formigas explorando.



Fonte: Foto do software criado durante execução

Na Figura 1, observa-se apenas o feromônio de caminho (*path*). Durante a simulação, no entanto, existem outros dois tipos de feromônio: o feromônio de comida (*food*), representado pela cor verde, e o feromônio de intruso (*alarm*), que aparece em vermelho no formigueiro azul e em azul no formigueiro vermelho.

O menu na parte inferior da imagem foi desenhado e funcionalmente implementado utilizando OpenGL, uma vez que essa API também permite interações com o usuário. O menu inclui um botão de *play/pause*, um botão de *reset* e três botões que ajustam o visual da simulação. O primeiro controla a exibição da simulação na tela, enquanto os outros dois determinam a visualização dos feromônios de cada formigueiro individualmente.

3.2 Paralelismo

Inicialmente, a busca por maior eficiência surgiu porque o desenho dos feromônios na tela estava desacelerando a simulação além do desejável. Em cada quadro, o programa percorre uma matriz de feromônios de 900x900; caso um feromônio esteja presente em determinada posição, um quadrado centrado no pixel é desenhado, representando o feromônio local por meio de uma cor específica. A solução, contudo, não envolveu paralelismo, pois, ao desenhar e preencher um quadrado, não é possível renderizar outro sem interferir no primeiro. Ambos compartilham uma lista de pontos a serem preenchidos, resultando em conflitos entre si.

O problema foi solucionado alterando o tipo de variável da matriz de feromônio para *unsigned char* e reduzindo o número de operações realizadas nessa parte do código. Essas otimizações melhoraram o desempenho da simulação, evitando o impacto negativo que o desenho dos feromônios exercia sobre a velocidade de execução.

Embora não tenha sido possível melhorar o desempenho ao desenhar na tela, foi possível otimizar outras partes do código. O foco principal das melhorias esteve nas operações realizadas nas matrizes de feromônio e nas demais matrizes que representam a tela, como aquelas que indicam a posição da comida e das formigas.

Embora não tenha sido possível otimizar o desempenho do desenho na tela com paralelismo, foi viável aprimorar outras partes do código. O principal foco de otimização foi reduzir o custo das operações realizadas na matriz de feromônio e em outras matrizes que representam a tela, como a matriz que registra a posição da comida e a que indica a localização das formigas. Essas melhorias reduziram o tempo de processamento em operações repetitivas e aumentaram a eficiência geral da simulação.

Código-fonte 1: Paralelismo de matrizes

```
1     #pragma omp parallel for
2     for(int i=0; i<900; i++)
3         for(int j=0; j<900; j++){
4             colony->ant_position[i][j] = false;
5             for(int k=0; k<3; k++)
6                 colony->pheromones[i][j][k]=0;
7         }
```

No Código-fonte 1 acima, observa-se uma das áreas do código que foi otimizada utilizando paralelismo. Nesta seção, é reiniciado a matriz que representa a posição das formigas e a matriz de feromônios, que contém três tipos de feromônios em cada posição. A primeira linha do código informa ao compilador que o *loop* subsequente pode ser dividido entre os núcleos disponíveis, otimizando assim o desempenho dessa parte do código. Essa abordagem permite que múltiplas operações sejam executadas simultaneamente.

Outra parte do código que foi otimizada, que não está diretamente relacionada a uma matriz, é o comportamento das formigas. A cada quadro, cada formiga toma várias decisões sobre onde se mover e qual feromônio liberar. Essa otimização é evidenciada no Código-fonte 2 abaixo. A implementação do paralelismo nesse contexto permite que o comportamento das formigas seja processado de maneira mais eficiente dividindo a tarefa entre os diferente núcleos disponíveis no computador.

Código-fonte 2: Paralelismo de comportamento

```
1 void process_colony(Colony *colony, char enemy_location[900][900],
2                                                          Food *food){
3     update_pheromones(colony);
4
5     #pragma omp parallel for // workers
6     for(int i=colony->soldiers_amount; i<colony->ants_amount; i++){
7         if(colony->ants[i].alive){
8             ant_behaviour(colony, &colony->ants[i],
9                           colony->pheromones, enemy_location, food);
10            move_ant(&colony->ants[i], colony->ant_position, true);
11        }
12    }
13    #pragma omp parallel for // soldiers
14    for(int i=0; i<colony->soldiers_amount; i++){
15        if(colony->ants[i].alive){
16            soldier_behaviour(colony, &colony->ants[i],
17                              colony->pheromones, enemy_location);
18            move_ant(&colony->ants[i], colony->ant_position, false);
19        }
20    }
21 }
```

Com as otimizações mencionadas, foi possível manter a simulação estável em 60 quadros por segundo. Essas melhorias no desempenho garantiram uma experiência mais fluida e responsiva, permitindo que a simulação funcionasse de maneira eficaz, mesmo com a complexidade das interações entre as formigas e a dinâmica dos feromônios.

3.3 Estrutura do Código

3.3.1 Formiga

Cada formiga simulada utiliza a estrutura apresentada no Código-fonte 3. Nessa estrutura, são armazenadas as posições em *x* e *y* da formiga, além da direção em que ela está orientada, representada pela variável *theta*. A variável *initial theta* armazena a direção inicial da formiga no início da simulação, servindo para restaurar a orientação original quando o botão de *reset* é acionado, o que garante que cada formiga possa ser adequadamente reiniciada, mantendo a consistência na simulação.

Código-fonte 3: Estrutura de Formiga

```
1 typedef struct _ant{
2     int  x, y, theta, initial_theta;
3     bool found_food, intruder_detected;
4     int home_sick, alarm_max;
5     int lost;
6     bool alive;
7 }Ant;
```

As variáveis *found food* e *intruder detected* são definidas como verdadeiras quando a formiga entra em contato com uma fonte de alimento ou com uma formiga rival, respectivamente. Essas variáveis desempenham um papel crucial no comportamento da formiga, permitindo que ela reaja adequadamente a situações importantes no ambiente, como a coleta de recursos e a defesa contra intrusos.

As variáveis *home sick* e *alarm max* são utilizadas para limitar a capacidade da formiga de liberar feromônio caso ela não encontre o ninho dentro de um tempo aceitável. A variável *home sick* indica que a formiga está com dificuldade para retornar ao ninho, enquanto *alarm max* define um limite de tempo máximo para a formiga depositar o feromônio de alarme, evitando que ela continue a marcar feromônios de forma ineficiente em situações em que a busca pelo ninho se prolonga excessivamente. Essas variáveis ajudam a manter o comportamento da formiga mais eficiente durante a simulação.

A variável *lost* é um contador responsável por indicar se a formiga se desviou do caminho e, portanto, se perdeu durante a simulação. Por fim, a variável *alive* serve para determinar se a formiga ainda está ativa; caso ela tenha sido morta durante a simulação, essa variável sinaliza para que não sejam realizados novos processos ou desenhos relacionados a essa formiga.

3.3.2 Colônia

O formigueiro utiliza um vetor composto pela estrutura das formigas, conforme descrito no subcapítulo anterior. As formigas são divididas em trabalhadoras e soldados, de acordo com a variável *soldier amount*, que define quantas das primeiras formigas no vetor assumem o papel de soldados. Além disso, a variável *ants amount* determina o número total de formigas no formigueiro. As variáveis têm tratamento para garantir que o número de soldados não ultrapasse o número total de formigas.

Cada formigueiro possui uma matriz de posições das suas formigas, utilizada pelo formigueiro rival para detectar e definir situações de conflito. Além disso, cada formigueiro mantém uma matriz de feromônios com três camadas, representando os diferentes tipos de feromônio: um para exploração, outro para indicar o caminho até a comida, e o terceiro para sinalizar a presença de invasores. Essa estrutura permite que cada colônia tenha um mapeamento detalhado de suas atividades.

Código-fonte 4: Estrutura de colônia

```
1 typedef struct _colony{
2     std::vector<Ant> ants;
3     char ant_position[900][900];
4     unsigned char pheromones[900][900][3];
5     bool draw_phero;
6     float nest_x, nest_y;
7     int ants_amount;
8     int soldiers_amount;
9
10    int home_sick_max;
11    int food_found_amount;
12
13    unsigned char alarm_phero_amount;
14    unsigned char food_phero_amount;
15    unsigned char path_phero_amount;
16
17    unsigned short int decay_timer, decay_timer_max;
18    unsigned char decay_amount;
19 }Colony;
```

A variável *draw phero* controla a exibição dos feromônios dessa colônia na tela. Conforme ilustrado na Figura 1, o botão de feromônio está associado a essa variável, permitindo que o usuário ative ou desative a visualização dos feromônios na simulação conforme desejado.

A quantidade de alimento colhido com sucesso é representado na variável *food found amount*.

A variável *home sick max* define o tempo máximo que uma formiga pode permanecer fora do ninho. Esse limite impede que a formiga fique vagando indefinidamente, promovendo um retorno ao ninho após um período específico e ajudando a manter o ciclo de atividades dentro do formigueiro. As variáveis de quantidade de feromônio definem a quantidade específica de cada tipo de feromônio que é depositado no ambiente sempre que necessário.

Por último, as variáveis relacionadas ao decaimento de feromônio definem a frequência com que o decaimento ocorre e a quantidade de feromônio removido a cada ciclo. Essas variáveis são essenciais para simular o envelhecimento e a dissipação natural dos feromônios no ambiente, garantindo que as trilhas de feromônio não permaneçam ativas indefinidamente e que o sistema continue a refletir mudanças dinâmicas no comportamento das formigas e no ambiente.

3.3.3 Processando Colônia

A cada ciclo de processamento, a simulação atualiza o estado das duas colônias, ajusta a posição da comida no mapa, caso necessário, e redesenha a janela visualizada pelo usuário. No Código-fonte abaixo, estão listadas as funções responsáveis pelo processamento interno, que são chamadas em cada ciclo.

Código-fonte 5: Processando colônias

```
1    process_colony(&blue_ants, red_ants.ant_position, &food);
2    process_colony(&red_ants, blue_ants.ant_position, &food);
3    process_food(&food);
```

A função *process food()* verifica se já foram removidas 25 unidades de comida da fonte de alimento. Se esse limite for atingido, a função reposiciona a fonte de alimento aleatoriamente em uma nova área: à esquerda e acima ou à direita e abaixo, dependendo de sua posição anterior. Essa mudança garante que as formigas não possam reutilizar o caminho previamente traçado até a fonte, forçando-as a explorar novamente o ambiente. Esse mecanismo aumenta a probabilidade de encontros entre formigas das colônias rivais, incentivando conflitos e promovendo interações competitivas na simulação.

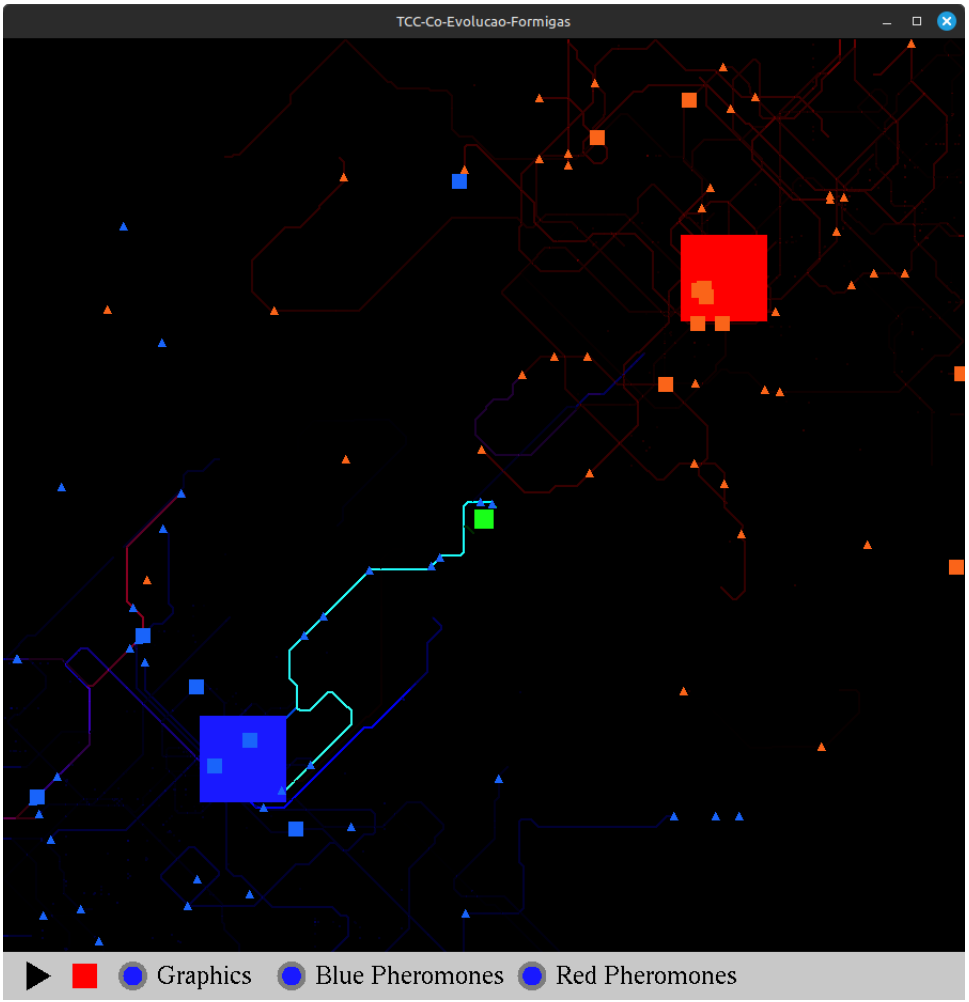
A função *process colony()*, visualizada no Código-fonte 2, realiza três tarefas distintas: atualiza o mapa de feromônios, processa o comportamento das formigas trabalhadoras e por último processa o comportamento das formigas soldados.

Na função de atualização dos feromônios, quando a variável responsável pelo decaimento de feromônio da colônia (*decay timer*) atinge zero, um *loop* percorre a matriz de feromônios, reduzindo a quantidade de feromônio em cada posição por um valor previamente definido (*decay amount*). Esse *loop* foi otimizado com paralelismo, acelerando o processo de decaimento e garantindo que a atualização do estado dos feromônios não tenha um custo alto de processamento.

Com a matriz de feromônios atualizada, inicia-se o processamento do comportamento de cada formiga da colônia. Nesse estágio, cada formiga avalia seu ambiente com base nos feromônios ao seu redor, na presença de alimento próximo, na presença de intrusos e no tempo em que a formiga esteve fora do formigueiro.

Inicialmente, se não houver fontes de alimento ou rivais presentes, a formiga deixa o formigueiro e começa a se mover aleatoriamente em busca de comida, depositando feromônio de caminho (*path*) para marcar sua rota. Ao encontrar uma fonte de alimento, a formiga inicia o retorno, seguindo o caminho previamente marcado, enquanto começa a soltar feromônio relacionado à comida (*food*). Se a formiga se perder durante o trajeto (indicada pela variável *lost*), ela interrompe a liberação do feromônio de comida. Caso a formiga não tenha encontrado a fonte de alimento após um longo período de busca (quando a variável *home sick* atinge zero), ela tenta retornar ao formigueiro. Além disso, caso encontre uma formiga rival, ela imediatamente começa a voltar para o ninho, liberando feromônio de alerta de intruso (*alarm*) para sinalizar a presença da ameaça, também parando de soltar feromônio se se perder.

Figura 2 – Simulador com formigas trazendo comida ao formigueiro.



Fonte: Tela do software obtida durante execução

Na Figura 2, é observado que as formigas azuis depositando feromônios de caminho (*path*) e de comida (*food*) ao longo do trajeto entre o formigueiro azul e a fonte de alimento, formando uma linha de cor verde-azulada que conecta os dois pontos

Por outro lado, as formigas soldado seguem constantemente os caminhos criados por outras formigas da mesma colônia, com o objetivo de encontrar formigas rivais ou rastros de feromônio de alerta. Quando uma formiga soldado encontra uma formiga rival trabalhadora, ela a elimina caso esteja sobre ou ao lado dela. Se a formiga soldado encontrar outra formiga soldado, ambas se enfrentam até colidirem na mesma posição, resultando na morte de ambas as formigas.

A função *move ant* atualiza a posição das formigas e trata de colisões com a borda do mapa.

Os comportamentos das formigas trabalhadoras e soldados orientam a próxima posição para a qual a formiga se moverá. Cada formiga possui uma posição registrada na matriz de posições do formigueiro. Quando a formiga está seguindo um rastro de feromônio, sua tendência é continuar na direção atual até que o feromônio à frente desapareça, incentivando-a a explorar outras áreas ou responder a estímulos próximos.

Código-fonte 6: Movimento da formiga

```
1 // try to follow a path, uses 'find_pheromone' if lost
2 bool follow pheromone( ... , int type){
3
4     if(pheromones[x][y][type] == 0) // not in pheromone path of type
5         return find_pheromone(x, y, theta, pheromones, type);
6
7     /*
8         G H A     . A B     A B C
9         F X B     . X C     H X D     ...
10        E D C     . . .     G F E
11
12        -->         ^ -->       ^
13
14        3 2 1
15        4 X 0  *theta
16        5 6 7
17    */
18    ...
19 }
```

A formiga tenta seguir constantemente o caminho do feromônio, e quando não está diretamente sobre ele, a função de busca pelo feromônio é acionada. No código acima, é visualizado a lógica utilizada para simular o comportamento da formiga ao seguir um rastro.

A direção da formiga é representada pela letra B. Caso o feromônio não continue à frente (na direção B), a formiga verifica as direções à sua esquerda (A) e à sua direita (C), escolhendo a direção que contém o maior rastro de feromônio. Se não houver rastros de feromônio nessas direções, a formiga examina suas laterais (H e D) e continua buscando até ter uma visão completa de seu entorno. Caso a formiga fique ilhada sem feromônios nas direções possíveis, ela aciona novamente a função de busca para tentar encontrar o caminho.

Quando uma formiga está perdida ou em busca de um feromônio específico, ela utiliza uma estratégia de visão para buscar o melhor caminho. Na representação abaixo, o X simboliza a posição da formiga, A representa a antena esquerda e B a antena direita. Os pontos (C) indicam o caminho em frente.

Durante esse processo, a formiga observa todas as posições à sua frente e avalia qual delas contém o maior rastro do feromônio desejado. Com base nessa análise, a formiga ajusta sua direção de movimento, escolhendo a direção que oferece o maior caminho de feromônio.

Código-fonte 7: Busca pelo feromônio

```
1  /*  X - ant
2      A - left sensor
3      B - right sensor
4                          A A A . B B B
5                B B B   A A A . B B B   A A A
6              . B B B     A A . B B     A A A .
7            A A . B             X             A . B B
8            A A A X                           X B B B
9            A A                                   B B
10
11         B B                                       A A
12         B B B                                   A A A
13         B B B                                   A A A
14         . . . X                               X . . .
15         A A A                                   B B B
16         A A A                                   B B B
17         A A                                       B B */
18     int sumA=0, sumB=0, sumC=0; // sum of pheromones in it's path
```

Quando uma formiga não encontra nenhum rastro do feromônio desejado, ela recorre ao andar aleatório até localizar um caminho com o feromônio. Esse movimento é controlado por uma probabilidade específica para decidir sua próxima direção. A cada passo, a formiga tem: 1 em 50 chance de virar à esquerda, 1 em 50 de chance de virar à direita e caso contrário, ela continua seguindo reto. Este comportamento é utilizado quando a formiga está explorando. Dessa forma, a formiga pode reagir assim que encontra uma fonte de alimento ou o feromônio que pode levar a uma.

Capítulo 4

EVOLUÇÃO E RESULTADOS

4.1 Método

Para promover a coevolução entre os dois formigueiros, são aplicados Sistemas Evolutivos, nos quais cada formigueiro é avaliado em relação ao desempenho do outro. A cada geração, o desempenho de cada colônia é comparado ao da colônia rival, incentivando uma adaptação constante para superar as estratégias e vantagens do adversário. Esse processo competitivo gera uma evolução mútua, em que cada geração tende a melhorar para enfrentar o melhor da geração rival.

As variáveis ajustadas para gerar cada indivíduo são as seguintes:

- *decay timer max*, *decay amount* : variáveis que definem o decaimento do feromônio no mapa.
- *alarm phero amount*, *food phero amount*, *path phero amount* : variáveis que definem a quantidade de feromônio depositado de cada tipo.
- *home sick max* : define a distância percorrida pela formiga antes de decidir voltar ao formigueiro.
- *soldiers amount* : define o número de formigas soldados, o número total de formigas está fixo em 50.

Para estabelecer a primeira geração de formigas, criam-se dois formigueiros: um azul e outro vermelho. O formigueiro vermelho é configurado com valores de variáveis que, segundo o autor, se aproximam das condições ideais para a simulação. Esses valores incluem: *decay timer max* = 16, *decay amount* = 1, *alarm* = 50, *food* = 100, *path* = 60, *home sick max* = 550 e *soldiers amount* = 10.

Para o formigueiro azul, um conjunto de variáveis, chamado *indivíduo*, é gerado de forma aleatória e testado em uma simulação contra o formigueiro vermelho. A simulação possui duas condições de parada: a primeira ocorre ao atingir o tempo limite (numero de ciclos de processamento limite equivalentes a rodar sem paralelismo por 1 minuto), indicando que o indivíduo não foi suficientemente eficiente; a segunda condição ocorre quando um dos formigueiros acumula 100 unidades de comida no ninho. Durante a simulação, a fonte de

alimento muda de posição a cada 25 unidades removidas, incentivando as formigas de ambos os formigueiros a explorar e se confrontar. Independentemente da condição de parada acionada, o indivíduo é considerado bem-sucedido se acumular mais de 70 unidades de alimento. A primeira geração de formigas azuis é composta pelos primeiros 10 indivíduos que atendem a esse critério de sucesso.

Após obter os 10 indivíduos do formigueiro azul, cada um é testado em 10 simulações contra o melhor do formigueiro vermelho. A média do tempo e da quantidade de comida coletada em cada rodada é calculada para avaliar melhor o desempenho dos indivíduos e reduzir o fator de sorte. O melhor indivíduo da geração é então escolhido com base na quantidade de comida coletada, respeitando as condições de parada estabelecidas. Em caso de empate na quantidade de comida, o indivíduo com o menor tempo médio é considerado o melhor. Nessa etapa de avaliação, o tempo limite é dobrado, proporcionando uma maior oportunidade para que o indivíduo alcance a quantidade desejada de alimento. Contudo, raramente o tempo máximo é alcançado.

Com o formigueiro azul definido, o processo é repetido para o formigueiro vermelho, agora utilizando o melhor indivíduo do formigueiro azul como rival na simulação. Dessa forma, a primeira geração de ambos os formigueiros é estabelecida.

A partir da segunda geração, novos indivíduos não são mais gerados aleatoriamente. Em vez disso, a nova geração é criada a partir dos indivíduos da geração anterior por meio de um processo de *crossover*. Nesse processo, o melhor indivíduo da geração anterior reproduz com os demais, sendo preservado inalterado, enquanto os outros passam por pequenas mutações em suas variáveis, aplicadas de acordo com uma taxa de mutação controlada.

Assim, o código segue um ciclo contínuo de otimização evolutiva: reprodução (*crossover* e mutação para gerar a nova geração), simulação e avaliação (executando a simulação para avaliar o desempenho de cada indivíduo), e seleção (escolhendo o melhor indivíduo com base nos critérios definidos). Em seguida, o mesmo processo é repetido para o formigueiro adversário, mantendo a coevolução entre as colônias enquanto cada uma busca constantemente superar a outra.

Com o objetivo de comparar diferentes simulações, foi decidido limitar a simulação a apenas 10 gerações.

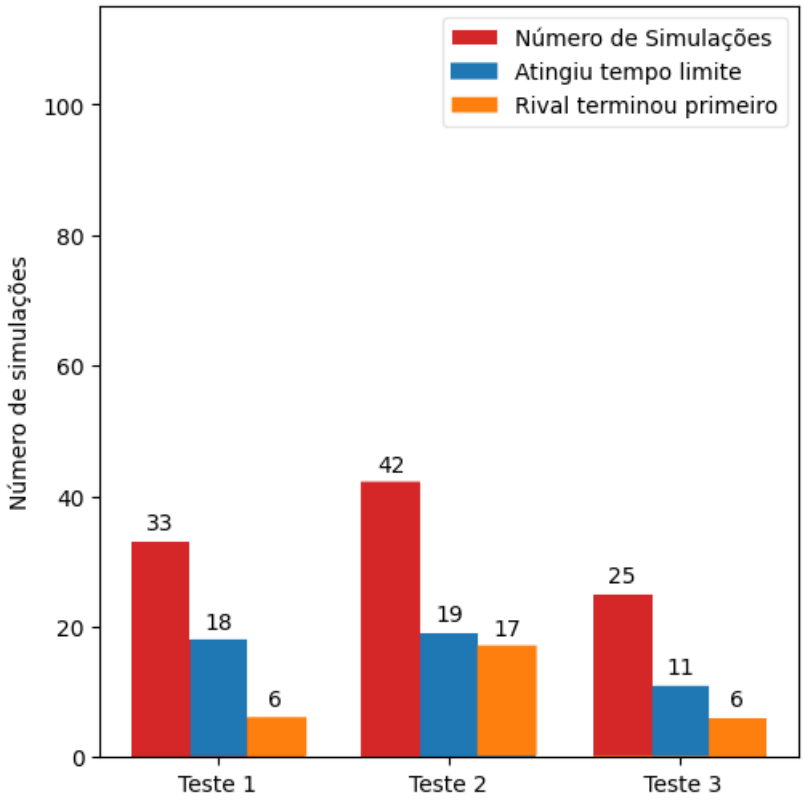
4.2 Resultados

4.2.1 Primeira geração - formigueiro azul

Durante a criação da primeira geração de indivíduos do formigueiro azul (conjuntos de variáveis que definem seu comportamento), observou-se que muitas simulações atingiram o limite de tempo antes que qualquer formigueiro pudesse alcançar a quantidade-alvo de alimento. Esse resultado era esperado, pois o limite de tempo foi implementado justamente para reduzir o processamento gasto em indivíduos com desempenho inferior. Dessa forma, apenas os indivíduos que demonstraram eficiência na coleta de alimento foram selecionados para compor a primeira geração, garantindo que somente as estratégias mais promissoras avançassem para a próxima fase da simulação.

Foram realizadas três simulações de 10 gerações cada, denominadas Teste 1, Teste 2 e Teste 3, para avaliar o comportamento dos resultados. A Figura 3 apresenta dados referentes à primeira geração do formigueiro azul. O gráfico mostra, na primeira barra, o número total de simulações realizadas, seguido pela quantidade de simulações que atingiram o tempo limite e, por fim, o número de simulações encerradas devido ao formigueiro rival ter coletado 100 unidades de alimento. Esses três conjuntos de dados são apresentados para cada um dos testes realizados.

Figura 3 – Gráfico da primeira geração azul de cada teste



Fonte: criação do autor

Nas três simulações realizadas, observou-se que foram necessárias cerca de 30 simulações para encontrar os 10 indivíduos necessários para compor a primeira geração. Aproximadamente metade dessas simulações excedeu o limite de tempo, enquanto um número reduzido de simulações foi concluído como sucesso do formigueiro rival, utilizando os parâmetros definidos manualmente.

Após a seleção dos 10 indivíduos que compõem a primeira geração, foram realizadas dez simulações com cada indivíduo para avaliar seus desempenhos, agora com tempo limite dobrado. Conforme esperado, a maioria dos indivíduos conseguiu coletar, em média, mais de 90 unidades de alimento. No entanto, ocasionalmente, um ou dois indivíduos apresentaram um desempenho inferior, coletando uma quantidade menor de alimento. Isso ocorre porque, embora possam ter tido sorte na seleção inicial, não eram realmente indivíduos com características vantajosas para a tarefa.

O melhor indivíduo do formigueiro azul é selecionado entre os mais bem-sucedidos da primeira geração. Considerando que essa geração geralmente atinge valores próximos à pontuação máxima, a escolha do melhor indivíduo costuma se basear no tempo de execução. Dessa forma, entre os indivíduos que atingiram a pontuação máxima, aquele que completou a tarefa em menor tempo é considerado o melhor.

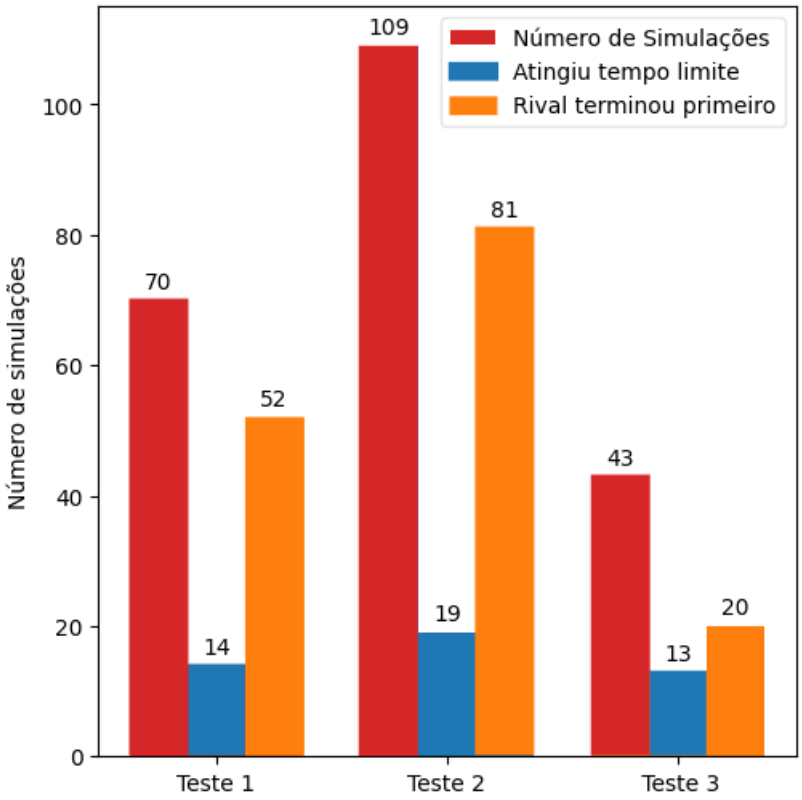
4.2.2 Primeira geração - formigueiro vermelho

Com o melhor indivíduo do formigueiro azul identificado, inicia-se a busca por 10 indivíduos do formigueiro vermelho, aplicando os mesmos critérios utilizados para a seleção dos indivíduos azuis. No entanto, nesta etapa, em vez de utilizar valores fixos como referência, o melhor indivíduo do formigueiro azul previamente encontrado é empregado como rival.

Ao utilizar como rival o melhor indivíduo do formigueiro azul, que se destacou por sua eficiência em localizar alimento entre todos os simulados até aquele ponto, a dificuldade de obtenção das 100 unidades de alimento aumentou significativamente. Isso ocorre porque as formigas azuis, agora otimizadas para completar a simulação de maneira eficiente, reduzem as oportunidades para as formigas vermelhas se destacarem. Dessa forma, a seleção dos indivíduos vermelhos para a primeira geração se torna mais rigorosa, exigindo maior competitividade e desempenho para serem considerados aptos.

Esse comportamento pode ser visualizado na Figura 4, onde o número de simulações totais e de simulações em que o formigueiro rival obteve sucesso primeiro é notavelmente alto. Em contrapartida, o número de simulações que ultrapassaram o tempo limite apresentou uma redução quando comparado ao número total de simulações realizadas.

Figura 4 – Gráfico da primeira geração vermelha de cada teste



Fonte: criação do autor

Durante a simulação de cada indivíduo do formigueiro vermelho por 10 vezes, os resultados observados foram semelhantes aos do formigueiro azul. A maioria dos indivíduos demonstrou um desempenho consistente, conseguindo finalizar as simulações com uma pontuação elevada. Isso reforça a eficácia do processo evolutivo, mostrando que, mesmo com o aumento da competitividade introduzida pelo formigueiro azul otimizado, as formigas vermelhas adaptaram-se bem e foram capazes de competir de forma eficiente.

4.2.3 Segunda geração

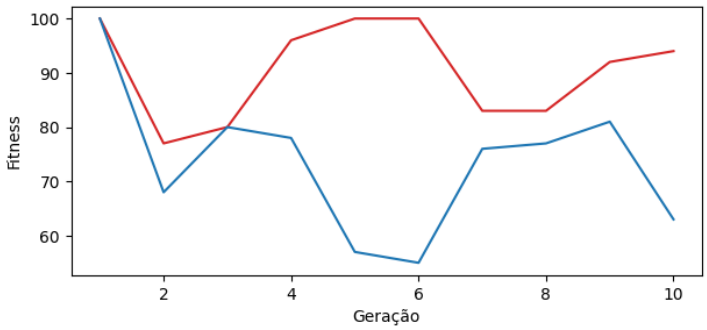
Na segunda geração, as formigas azuis passam pelo primeiro ciclo de reprodução. Com a evolução das formigas vermelhas, que foram especificamente otimizadas para competir de forma eficaz contra as azuis, observa-se uma queda significativa no desempenho do formigueiro azul. Esse processo reflete a natureza dinâmica da coevolução, na qual os avanços em uma colônia impulsionam adaptações na outra.

Após a evolução do formigueiro azul, o desempenho do formigueiro vermelho também tende a diminuir em eficácia, refletindo a pressão seletiva imposta pela melhoria de seus rivais. No entanto, devido à elevada qualidade inicial dos indivíduos vermelhos, a pontuação média dos membros da nova geração ainda se mantém mais alta.

4.2.4 Gerações subsequentes

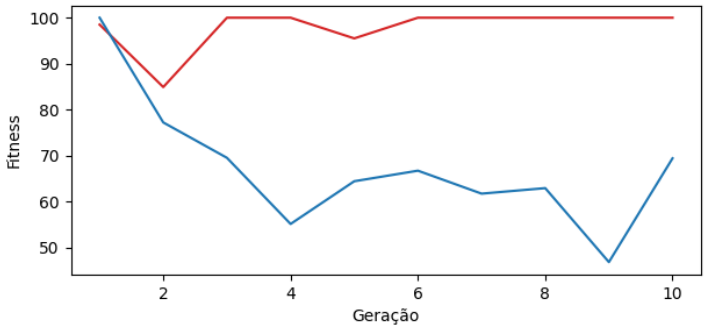
A partir da terceira geração, nota-se um padrão semelhante entre as simulações realizadas. Como ilustrado nas Figuras 5, 6 e 7, observa-se que os formigueiros vermelhos geralmente mantêm uma vantagem sobre os azuis. No entanto, considerando a natureza caótica inerente aos sistemas evolutivos, e em especial analisando a simulação representada na Figura 5, pode-se inferir que o formigueiro azul tem potencial de superar o formigueiro vermelho com o passar do tempo. Os gráficos a seguir representam as unidades de alimento pegas pelos melhores de cada geração em cada formigueiro.

Figura 5 – Gráfico do primeiro teste



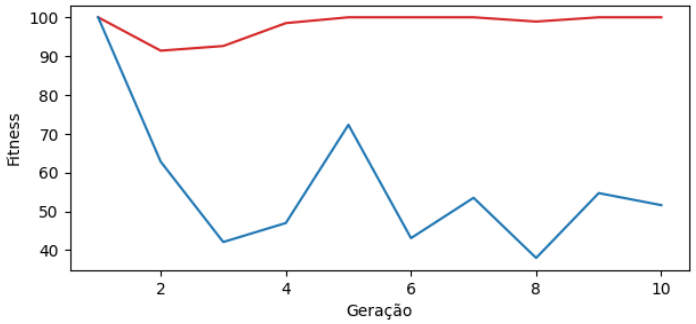
Fonte: criação do autor

Figura 6 – Gráfico do segundo teste



Fonte: criação do autor

Figura 7 – Gráfico do terceiro teste



Fonte: criação do autor

Capítulo 5

CONCLUSÃO

O software desenvolvido neste trabalho teve como objetivo principal simular a coevolução de duas colônias de formigas, e esse objetivo foi alcançado com êxito. Os resultados obtidos estiveram de acordo com as expectativas iniciais. Desde a primeira geração, os sistemas evolutivos e a coevolução foram evidenciados, com resultados significativamente superiores em relação aos parâmetros definidos manualmente. Por meio da implementação e análise dos formigueiros em competição, foi possível observar claramente os processos de adaptação e evolução, confirmando a eficácia da abordagem adotada para investigar a dinâmica de sistemas evolutivos e coevolutivos.

Durante o desenvolvimento deste trabalho, o entendimento sobre o comportamento das formigas e os diversos tipos de insetos que utilizam feromônios foi significativamente aprofundado. Esse conhecimento foi fundamental para a criação de uma simulação mais realista e precisa, contribuindo substancialmente para o sucesso do projeto e para a modelagem adequada dos processos de comunicação e interação entre as colônias de formigas.

Ao longo da simulação dos formigueiros, observou-se que o processo de reprodução dos indivíduos não foi eficiente nas primeiras gerações. Isso ocorreu porque aproximar um indivíduo aleatório do melhor indivíduo não garantiu uma melhoria significativa, o que resultou em uma perda de eficiência nas gerações iniciais. No entanto, esse problema foi minimizado nas gerações subsequentes, pois os indivíduos já estavam mais próximos uns dos outros, mostrando que o método de reprodução é eficaz para os indivíduos de gerações maiores.

Como trabalhos futuros é sugerido alterar o método de reprodução e otimizar o tempo de execução da simulação são potenciais áreas para melhorias. A modificação do processo de reprodução poderia ajudar a acelerar a evolução e aumentar a eficiência desde as primeiras gerações. Além disso, otimizar o tempo de simulação permitiria realizar experimentos mais complexos e com um número maior de gerações, oferecendo uma análise mais aprofundada do comportamento evolutivo dos formigueiros.

Neste trabalho, houve um aprofundamento significativo no conhecimento sobre Sistemas Evolutivos e paralelismo, temas amplamente utilizados e explorados durante o desenvolvimento. Ambos os conteúdos foram abordados ao longo do curso, mas seria interessante que houvesse mais oportunidades em disciplinas que aplicassem e reforçassem tais conceitos.

REFERÊNCIAS

GABRIEL, P. H. R.; DELBEM, A. C. B. **Fundamentos de algoritmos evolutivos**. [S.l.]: ICMC-USP, 2008. Citado 2 vezes nas páginas 15 e 18.

LORENCI, A. D. **Pray-Predator Coevolution**. 2020. Disponível em: <https://github.com/AliceDeLorenci/pray-predator-coevolution>. Citado na página 15.

RESH, V. H. **Encyclopedia of Insects**. 2. ed. [S.l.]: Elsevier Science Publishing Co Inc, 2009. Citado 2 vezes nas páginas 15 e 17.

SILVA, M. L. O. da. **Implementação de um Sistema de Busca e Resgate Baseado em Inteligência de Grandes Enxames Acelerados em Hardware**. 2022. Disponível em: <https://docs.google.com/document/d/1qk4VQJ4Zoe5vtX5szoPPX5S1XCqfWsiE1M2cZvL6-f8/edit?tab=t.0>. Citado 3 vezes nas páginas 15, 17 e 21.

WOO, M.; NEIDER, J.; DAVIS, T.; SHREINER, D. **OpenGL programming guide: the official guide to learning OpenGL, version 1.2**. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1999. Citado 2 vezes nas páginas 16 e 21.