UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Aplicação de algoritmo genético em colônias de formigas virtuais buscando encontrar automaticamente o comportamento de forrageamento.

Luís Fernando Estrozi

São Carlos - SP

Aplicação de algoritmo genético em colônias de formigas virtuais buscando encontrar automaticamente o comportamento de forrageamento.

Luís Fernando Estrozi

Supervisor: ***Eduardo do Valle Simões***

Monografia final de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP - para obtenção do título de Bacharel em Ciência da Computação

Área de Concentração: Robótica

USP - São Carlos

Junho de 2007

Sumário

Lista de Figuras

Resumo

Este projeto consiste na aplicação de um algoritmo evolutivo para otimizar o circuito de controle de agentes em um ambiente virtual para simulação de uma colônia de formigas buscando fontes de alimento. Os agentes ou formigas possuem um conjunto de parâmetros que configuram seu comportamento, como “probabilidade de sair de sua trajetória atual e virar para um determinado lado”, “quantidade de feromônio deixado quando se está buscando comida”, “nível de evaporação do feromônio”, dentre outros. Apesar de cada formiga ter um comportamento simples, o sistema como um todo, pode alcançar um objetivo comum bastante complexo. Utilizou-se um Algoritmo Evolutivo que foi executado em um sistema distribuído para buscar soluções que otimizassem os parâmetros das formigas, como saturação de feromônios, probabilidade de virar, etc., a fim de produzir automaticamente comportamentos exploratórios capazes de percorrer o ambiente em busca de fontes de alimento, gerando e otimizando um caminho entre estas fontes e o ninho. Quando encontradas estas fontes, os agentes devem retornar ao ninho marcando o caminho, representando um comportamento de forrageamento, no qual uma trilha de feromônio deve ser produzida entre o ninho e a fonte de alimento para atrair e guiar a maioria dos agentes para que coletem a maior quantidade de comida possível em um determinado tempo. Essa é uma tarefa muito difícil de ser obtida através da codificação do conhecimento de um especialista, pois este não pode configurar diretamente o comportamento do formigueiro e somente tem acesso à configuração dos comportamentos individuais dos agentes. Assim, o algoritmo evolutivo proposto neste trabalho traz uma contribuição significativa, pois possibilitou em algumas horas de execução a emergência de soluções melhores do que as implementadas pelos especialistas.

1. Introdução

Os sistemas que demonstram a emergência de comportamentos complexos a partir da interação de indivíduos simples vem sendo propostos na literatura já há algum tempo. O Jogo da Vida (Game of Life), elaborado pelo matemático britânico John Horton Conway, é descrito como uma matriz de células que podem estar em dois estados: *vivas* ou *mortas* (normalmente representados por apagado ou pintado, respectivamente). Cada célula interage com suas oito células vizinhas para definir o estado do sistema na próxima interação, sendo que ocorrem duas transições [1]:

1) Toda célula *morre* se tiver menos de duas ou mais de três vizinhas *vivas*.
2) Toda célula *morta* com exatamente três vizinhos torna-se *viva*.

Apesar da simplicidade de suas regras e do comportamento individual de cada célula, o sistema como um todo é complexo [2], podendo-se notar grupos de células estáveis, como o *glider*, que se move pela "matriz". Há também sistemas muito mais complexos, como o *Gosper glider gun*, que "produz" *gliders* a cada trinta gerações, lançando-os na matriz.

Esse mesmo tipo de "individualidade simples, com inteligência coletiva" também pode ser notado na natureza, observando-se o comportamento de diversos animais como, por exemplo, pássaros, peixes e formigas [3].

Abordagens simples para solução de problemas complexos, quando existem, são comumente interessantes, necessitam de poucos sensores, hardware simples e possuem comportamento simples. Entretanto, no sistema como um todo, pode ser notado o chamado *comportamento emergente*, no qual a qualidade da solução resultante é maior que a soma das partes [4].

Há diversas razões para o interesse no desenvolvimento de sistemas que fazem uso de times de agentes simples com comportamento coletivo emergente [5], entre elas:

- Algumas tarefas podem ser muito complexas (ou impossíveis) para serem resolvidas por um único agente; ou a performance pode ser melhor quando são utilizados múltiplos agentes;
- Construir e usar vários agentes simples pode ser mais fácil, flexível, ter menor custo e ser

mais tolerante a falhas do que ter um único agente complexo para cada tarefa separada;
- A abordagem de sistemas cooperativos pode elucidar problemas fundamentais das ciências sociais (teoria da organização, economia, psicologia cognitiva) e ciências da vida (biologia teórica, etologia animal);
- A grande capacidade de locomoção e exploração normalmente resultante;
- Escalabilidade inerente – A introdução de novos módulos para aumentar a capacidade/velocidade do sistema pode ser feita facilmente.

Apesar das vantagens descritas, a indústria não tem se voltado para a produção de sistemas multirobóticos. O motivo pode estar na dificuldade de se controlar um sistema multirobótico [6]. As técnicas de controle mais utilizadas para sistemas robóticos são sistemas de controle centralizados, nos quais a solução para o problema é programada com base no conhecimento de especialistas, gerando soluções rígidas de difícil adaptatividade às variações nas condições do ambiente de trabalho [7].

O universo de possibilidades que um robô pode encontrar no ambiente geralmente é bem maior do que a capacidade de previsão e antecipação dos projetistas. Quando, ao invés de um único robô, times de robôs são submetidos a ambientes reais, a quantidade de efeitos imprevisíveis que o sistema deve tratar cresce rapidamente, pois será maior o número de mecanismos e dispositivos interagindo com o meio [8].

A utilização de controles distribuídos, apesar da programação mais complexa, já que é necessário controlar concorrência de procedimentos, comunicação, etc., possui como vantagens a escalabilidade, tolerância a falhas e, normalmente, baixo custo das partes do sistema [9].

Inspirado pelo comportamento cooperativo das formigas naturais, mais especificamente a maneira como elas localizam o caminho mais curto da fonte de comida até o ninho, no começo dos anos 90 na Itália, Dorigo propôs em sua tese de doutorado os Algoritmos de Otimização baseados em colônias de formigas como uma técnica multiagente para resolução de problemas de otimização combinatorial.

Os algoritmos de otimização de formigas têm sido aplicados com sucesso para solucionar diversos problemas de otimização combinatória, como por exemplo, o do Caixeiro Viajante [10], de rotas de veículos, ordenação seqüencial, coloração de grafos,

rotas em rede de comunicação, o planejamento de compras [11], entre outros. Para resolver estes problemas com os algoritmos de formiga é requerido reduzi-los à busca pelo caminho mais curto em algum grafo, definir mecanismos de inicialização e atualização do feromônio e determinar regras heurísticas para a seleção de rotas [12].

Entre as principais características do Ant Colony Optimization advindas de formigas reais pode-se citar:

- um enxame de indivíduos cooperativos;
- uma trilha de feromônio (artificial), para comunicação estigmergética;
- uma política de decisão estocástica que faz uso de informação local.

O algoritmo busca um bom equilíbrio entre a solução ótima e o tempo de otimização e também pode ser aplicado a problemas de otimização combinatória estocástica [12].

Apesar do sucesso da aplicação desse algoritmo para solução dos problemas citados, pode-se verificar que normalmente as soluções encontradas não são tão boas, ou demoram mais para serem encontradas do que outros algoritmos desenvolvidos para aplicações específicas, como, por exemplo, algoritmos de minimização de caminhos em grafos [13].

Entretanto, esse algoritmo possui um bom desempenho em aplicações nas quais apenas parte do problema é conhecida, como neste trabalho, onde cada formiga irá basear suas decisões somente nos dados obtidos por seus sensores, que são locais.

Os algoritmos baseados em colônias de formigas são geralmente configurados por um grande número de parâmetros que têm de ser ajustados por um especialista para cada aplicação específica. Isso normalmente resulta em um trabalho considerável em um processo realimentado de tentativa-e-erro que nem sempre atende bem às necessidades do problema em questão. Assim, este trabalho vem propor a utilização de um algoritmo evolutivo para realizar automaticamente este ajuste, de maneira que não seja necessária a atuação de um especialista.

1.1. Objetivo

Este projeto consiste em construir um ambiente virtual que simule um ninho de formigas e fontes de alimento e projetar um Algoritmo Evolutivo para otimizar os

comportamentos individuais das formigas em função de uma tarefa global. O objetivo deste Algoritmo Evolutivo é procurar automaticamente um conjunto de parâmetros adequados para configurar o controlador de navegação de cada formiga artificial para que consigam encontrar objetos e trazê-los de volta ao ninho de maneira eficiente.

As formigas possuem diversos parâmetros como “probabilidade de sair de sua trajetória atual e virar para um determinado lado”, “quantidade de feromônio deixado quando se está buscando comida”, “quantidade de feromônio deixado quando se encontrou comida”, “nível de evaporação do feromônio”, “probabilidade de seguir feromônio quando se está buscando comida e quando já possui comida” e “tamanho do 'sensor' para detecção de alimentos ou feromônios”. A partir deste projeto, espera-se viabilizar em trabalhos futuros a construção de robôs reais cujos controladores de navegação utilizem as melhores características desenvolvidas no simulador proposto.

2. Síntese das Técnicas Envolvidas

2.1. Algoritmo Baseado em Formigas

As formigas são consideradas insetos sociais, pois vive em comunidade com outras formigas e possuem comportamentos mais voltados à preservação da sobrevivência da colônia como um todo do que a sua própria existência. Os insetos sociais, como formigas, cupins, abelhas e vespas, chamaram a atenção de muitos cientistas devido ao alto nível estrutural que suas colônias conseguem atingir, especialmente quando comparado com a relativa simplicidade dos indivíduos componentes da colônia [14].

Os insetos sociais possuem uma das estratégias de sobrevivência mais bem sucedidas da natureza, motivo de sua imensa quantidade e variedade: existem mais espécies de formigas em 1 quilômetro quadrado de uma floresta brasileira do que todas as espécies de primatas existentes no mundo [15].

Tal como nos bandos de aves ou cardumes de peixes, nas colônias de formigas não existe um líder que determina o comportamento individual de cada elemento da população com um objetivo bem determinado. Esse objetivo resulta da conjugação e interação entre os

elementos da população de forma espontânea. No entanto, ao contrário dos bandos de aves e cardumes de peixes, existe nas colônias de formigas uma sofisticação adicional: a realimentação positiva (*positive feedback*). Por exemplo, uma situação em que um organismo ou sistema químico produz uma enzima cuja presença incentiva a produção de mais desta mesma enzima.

Enquanto caminha da fonte de comida ao ninho e vice-versa, uma formiga deposita no chão uma substância chamada feromônio, formando em seu caminho uma trilha de feromônios, que se evapora ao longo do tempo. Os feromônios, por sua vez, podem ser facilmente detectados, através do olfato, pelas outras formigas. Assim, a trilha de feromônio permite a uma formiga encontrar o caminho de volta à fonte de comida ou ao ninho e, além disso, também permite que outras formigas localizem a fonte de comida encontrada [16]. A intensidade do feromônio em um local é maior quando alguma formiga passou recentemente por ali ou quando um grande número de formigas passou sobre o local.

As formigas tendem a escolher, com maior probabilidade, caminhos marcados por fortes concentrações de feromônio quando estão seguindo uma trilha de comida. Ao seguir uma trilha existente, as formigas tenderão a se concentrar nela, pelo simples fato de que a densidade de feromônio aumenta a cada formiga adicional que segue a trilha. Através deste mecanismo indireto de informação (realimentação positiva) as formigas conseguem colaborar na coleta de alimentos de uma dada fonte de comida [17].

Dorigo e seus colegas [10] basearam seus experimentos no fato de que ao caminhar do ninho para o alimento e vice-versa pelo menor caminho, as formigas retornarão mais rapidamente e, assim, passarão pelos mesmos pontos mais freqüentemente. Depositando assim uma trilha de feromônio mais densa. Quanto mais formigas optarem pela trilha densa, mais ela será reforçada. Em sua adaptação computacional desses comportamentos, foi deixada uma população de formigas buscar um mapa para o Caixeiro Viajante, aumentando a probabilidade de seguir uma conexão entre duas cidades como uma função do número de outras formigas simuladas que já seguiram aquela conexão. Pela exploração do efeito da realimentação positiva, ou seja, do reforço da trilha com cada formiga adicional, este algoritmo é capaz de resolver problemas combinatórios complexos, em que o objetivo é encontrar um meio de executar uma tarefa no menor número de passos

possível.

Assim, a idéia básica de Ant Colony Optimization é a de que um grande número de agentes simples e artificiais é capaz de construir boas soluções para problemas de otimização combinatórios difíceis via comunicações de baixo nível. Se em um dado momento uma formiga tenha que optar por diferentes caminhos, aqueles que foram mais vezes escolhidos previamente por outras formigas são escolhidos com maior probabilidade, já que terão maior quantidade de feromônios. As colônias de formigas artificiais buscam simular esse comportamento para obter um padrão de otimização [18].

2.2. Algoritmos Evolutivos

A Computação Evolutiva consiste em procedimentos de busca e otimização inspirados na Teoria de Evolução das Espécies de Darwin, tentando abstrair e imitar algumas das características da evolução natural com o propósito de encontrar soluções para problemas que requerem busca, adaptação e otimização [19].

Os paradigmas de Computação Evolutiva geralmente diferem dos paradigmas tradicionais de busca e otimização em três principais pontos:

- Utilizam uma população de pontos (soluções potenciais) em sua busca.
- Utilizam informação direta das funções a serem optimizadas, ao invés de derivações das funções ou outras técnicas relacionadas.
- Utilizam regras de transição de estados probabilísticas ao invés de determinísticas.

A maioria dos paradigmas tradicionais de busca e otimização consiste em se mover um ponto de um local a outro no espaço de decisão, utilizando alguma regra determinística [19]. Uma das desvantagens dessa abordagem é a possibilidade da busca ficar presa em um ótimo local. Esse comportamento normalmente é evitado pois os paradigmas de Computação Evolutiva utilizam uma população de pontos, o que possibilita a exploração simultânea de várias regiões ao mesmo tempo, reduzindo, assim, a probabilidade de ficar preso em um ótimo local. Operadores como crossover e mutação aumentam efetivamente essa capacidade de busca paralela, permitindo a busca ir diretamente de uma região

promissora do espaço para outra.

A estrutura básica de um Algoritmo Evolutivo é similar ao procedimento apresentado na Figura 1. A população geralmente é iniciada de forma aleatória e o critério de parada do algoritmo normalmente é baseado em se obter um indivíduo com alguma pontuação especificada, número de gerações, recursos disponíveis ou análise da variância dos resultados obtidos. Para a seleção de indivíduos para a reprodução existem vários métodos, como o da roleta, torneio e outros. Selecionados os pais, geram-se os filhos que irão compor a nova geração com o auxílio de operadores evolutivos, como crossover e mutação, que são os mais comuns [19].

1. Inicie a população.
2. Repita até o critério de parada ser satisfeito:
3. Calcule a pontuação de cada indivíduo na população.
4. Selecione indivíduos para a reprodução.
5. Gere novos indivíduos utilizando operadores evolutivos.
6. Escolha e separe os indivíduos para compor a nova população.

Figura 1. Estrutura básica de um Algoritmo Evolutivo [19]

2.3. Trabalhos Relacionados

Para o desenvolvimento deste trabalho, serão analisadas algumas plataformas similares existentes, buscando-se encontrar idéias que contribuam com a implementação dos algoritmos propostos.

Swarm – O projeto Swarm foi desenvolvido buscando oferecer uma linguagem e conjunto de rotinas para desenvolvimento e condução de experimentos em modelos baseados em agentes [20].

NetLogo – O NetLogo foi projetado especificamente para simulação de agentes móveis agindo concorrentemente em uma grade com comportamento dominado por interações locais em intervalos de tempos curtos. Embora modelos desse tipo são fáceis de serem implementados em NetLogo, a plataforma não é limitada a eles. O NetLogo é considerado a plataforma mais profissional em sua aparência e documentação [21].

The Ants: A Community of Microrobots – The Ants é uma comunidade de minirobôs simulados, cuja idéia foi obter uma comunidade robótica estruturada através da interação entre robôs, com inspiração em colônias de formigas [22].

iRobot Swarm – O projeto iRobot Swarm tem como objetivo desenvolver algoritmos distribuídos para comunidades de robôs compostas de centenas de indivíduos. A programação para cada robô deve ser robusta o suficiente para funcionar em ambientes reais [23].

O projeto Formiga [24], desenvolvido na tese de doutorado do aluno Danilo Nogueira Costa, no grupo de Sistemas Distribuídos do ICMC-USP, foi inspiração para este trabalho. Este sistema foi implementado em Java com a intenção de ser uma ferramenta facilmente adaptável para realizar testes com diferentes metodologias baseadas em inteligência de enxames e colônias de formigas artificiais. A escolha da linguagem Java vem facilitar a utilização da ferramenta por um grande número de usuários, mas por outro lado dificulta a geração de código eficiente e rápido.

Para melhorar a performance do simulador, este projeto partiu da tradução para linguagem C do projeto Formiga, no qual foram incluídos novos parâmetros e um Algoritmo Evolutivo para obtenção automática de soluções otimizadas para diversos ambientes. O software desenvolvido teve também como objetivo a velocidade de processamento, já que, para a execução do Algoritmo Evolutivo, seria necessária a avaliação de diversos casos de teste. O resultado obtido foi até oito vezes mais rápido que o projeto Formiga em condições similares.

A vantagem da solução proposta em relação às encontradas na literatura baseia-se na

simplicidade dos indivíduos, que podem ser programados em robôs de baixo custo, com poucos sensores. Também não há necessidade de um sistema de localização para a posição dos indivíduos ou de conhecimento da localização do ninho, já que as interações dependem apenas das concentrações de feromônio medidas pelos sensores.

3. Um Simulador de Colônias de Formigas Evolutivas

3.1. Plataforma de Desenvolvimento

Como linguagem de programação foi escolhida a linguagem C, por permitir que um programa desenvolvido adequadamente possa ter um desempenho elevado, o que é essencial para um simulador como o proposto devido à quantidade de iterações necessárias em cada simulação e complexidade das mesmas.

Para desenvolvimento da interface gráfica, foi escolhido o Borland C++ Builder, devido especialmente à sua facilidade de programação para ambiente gráfico, já que o desempenho não é crucial nesta versão, que deve servir apenas para a visualização do ambiente, uma vez que o processo evolutivo será realizado em background, sem necessitar de acompanhamento pelo usuário.

A versão para execução do Algoritmo Evolutivo foi compilada com o compilador GCC (GNU Compiler Collection) na plataforma Linux, não possui interface gráfica e utiliza a biblioteca PVM3 (Parallel Virtual Machine) [25] para execução do Algoritmo Evolutivo em um cluster, a fim de agilizar a obtenção dos resultados dos experimentos. A biblioteca PVM3 implementa a comunicação entre computadores conectados, fazendo o roteamento transparente de mensagens, conversão de dados de arquiteturas incompatíveis e outras tarefas necessárias para operação em um ambiente de computadores heterogêneos.

3.2. Implementação do processamento distribuído com PVM3

O sistema implementado baseia-se na interação entre *mestre* e *escravos*, da seguinte

forma:

1. O *mestre* cria a quantidade de escravos especificada e gera a população da primeira geração do Algoritmo Evolutivo de forma aleatória ou lendo um arquivo salvo anteriormente, caso tenha sido interrompido;
2. O *mestre* envia uma configuração de avaliação para cada *escravo*, sendo que são informados todos os parâmetros de configurações, mas não é informada a configuração do ambiente (posição dos obstáculos, ninho, etc.). Esta configuração é programada no próprio *escravo*, a fim de diminuir a quantidade de comunicação necessária, já que somente os parâmetros do Algoritmo Evolutivo são modificados durante o experimento e, portanto, precisam ser informados pelo *mestre;*
3. Ao receber uma configuração do *mestre*, o *escravo* faz a avaliação da mesma pela função de avaliação e envia uma mensagem ao *mestre* com o resultado obtido;
4. Ao receber uma resposta de um *escravo*, o *mestre* guarda-a em uma variável e envia uma nova configuração, caso a população da geração ainda não tenha sido totalmente avaliada;
5. Caso todos os filhos da população já tenham sido avaliados, o *mestre* aguarda até que o último *escravo* envie a nota do filho que está sendo avaliado;
6. Assim que todos os *escravos* finalizam suas tarefas, o mestre aplica o Algoritmo Evolutivo para calcular a próxima geração de filhos.

A quantidade de escravos foi determinada empiricamente, buscando-se minimizar o tempo de avaliação de cada geração, chegando-se ao valor de 58 *escravos* no cluster no qual foi executado, que possui 14 nós com processador Intel(R) Pentium(R) 4 CPU 3.40GHz dual e sistema operacional Linux.

Buscando-se reduzir o tempo de espera até que o último *escravo* enviasse a nota, diminuiu-se a relação da computação feita pela quantidade de comunicação (granularidade), de forma que apenas uma configuração é enviada aos *escravos* a cada iteração, mesmo em casos onde mais de um teste é efetuado em mais de um ambiente.

Com a configuração efetuada, o tempo de execução foi 15 vezes menor do que quanto executado em apenas um processador.

3.3. Implementação do Algoritmo Evolutivo

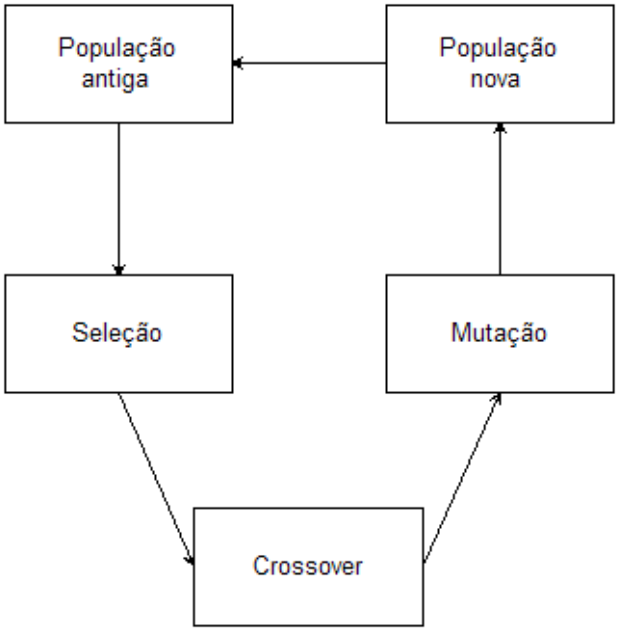

Figura 2. Fluxograma do Algoritmo Evolutivo implementado.

O Algoritmo Evolutivo implementado (fluxograma na Figura 2) possui, como cromossomo, o conjunto dos parâmetros a serem otimizados. Cada gene é um parâmetro específico, como saturação de feromônio, ângulo de virada, etc. Os seguintes parâmetros foram configurados: para o Algoritmo Evolutivo:

- Tamanho da população: quantidade de indivíduos a serem analisados em cada geração. A população inicial pode ser carregada de um arquivo ou gerada aleatoriamente.
- Taxa de mutação: Probabilidade que cada gene tem de sofrer uma variação respeitando o parâmetro “Porcentagem máxima da variação no gene quando a ocorrer mutação”. Por exemplo, caso a taxa de mutação de cada gene seja de 1% e a porcentagem máxima de variação no gene seja de 50%, espera-se que em 1% das reproduções, cada gene tenha seu valor aumentado ou diminuído entre 0 e 25% de seu valor. Se o valor do gene era 20, ele poderá passar a ter qualquer valor entre 10 e 30, desde que respeite os valores máximos e mínimos de cada gene.
- Porcentagem máxima da variação no gene quando a ocorrer mutação
- Taxa de crossover: É a taxa com que são trocados os genes dos pais para produzir o novo

filho. Quando ocorre crossover, o filho pode possuir o gene de um dos pais em 50% dos casos e a média deles em 50% dos casos.

- Quantidade de filhos aleatórios inseridos a cada {N} gerações: Esta configuração faz com que novos filhos sejam inseridos, no lugar dos filhos com as piores pontuações, depois de determinado número de gerações. O objetivo deste parâmetro é aumentar a variabilidade dos parâmetros, buscando evitar-se que máximos locais sejam encontrados.
- Quantos dos melhores filhos serão salvos para a próxima geração sem modificação: Com este parâmetro, busca-se preservar, para a próxima geração, os filhos com melhor pontuação, sem que esses sofram qualquer operação.

A seleção dos indivíduos que se reproduzirão é feita pelo método do torneio, pelo qual escolhe-se dois pares de indivíduos aleatoriamente e os melhores indivíduos de cada par escolhido serão os pais.

Para identificar os melhores parâmetros do Algoritmo Evolutivo, foi utilizado inicialmente um conjunto de 10000 testes, com diversa combinação de parâmetros e, posteriormente, outros 10000 testes, buscando-se refinar os parâmetros mais freqüentes encontrados pelo primeiro teste.

Os testes foram feitos com a algébrica Griewank Function [26] com a mesma quantidade de variáveis e 10 populações iniciais aleatórias. Os conjuntos de parâmetros que conseguiram fazer com que a média dos indivíduos ultrapassasse 90% do valor máximo da função foram selecionados para serem testados com outras 3 funções. A melhor configuração encontrada, baseada na quantidade máxima de chamadas à função de avaliação (determinada pelo tempo de processamento disponível) e com pontuação superior a 90% do máximo possível, foi:

- Filhos por geração: 90
- Taxa de mutação: 0,4%
- Porcentagem máxima da variação no parâmetro quando a mutação ocorrer: 60%
- Taxa de crossover: 4%
- Quantidade de filhos aleatórios inseridos a cada N (configurável) gerações: 13 filhos aleatórios a cada 3 gerações
- Quantos dos melhores filhos serão salvos para a próxima geração sem modificação: 1

3.4. Implementação da colônia de formigas

3.4.1. A interface gráfica

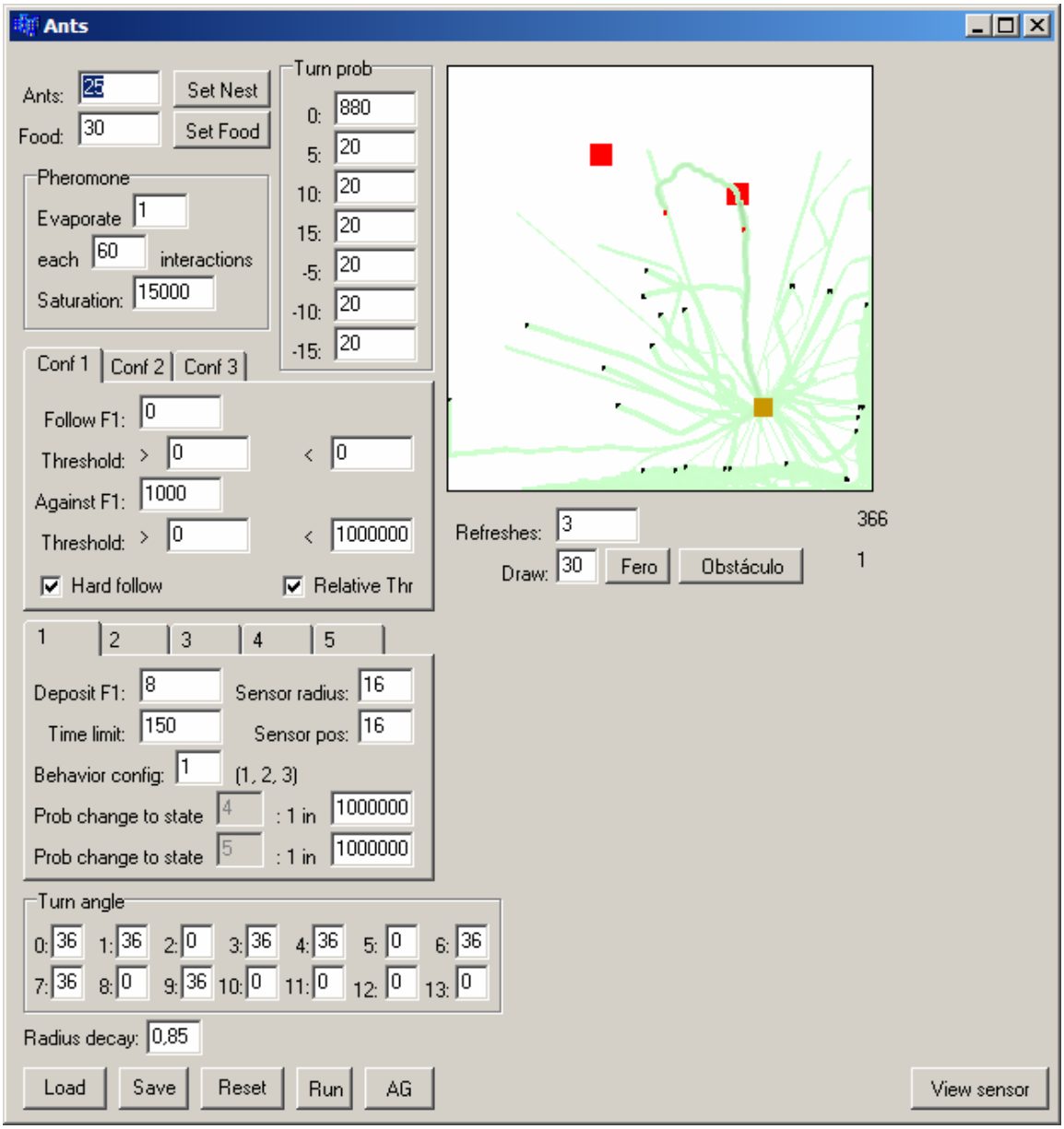

Figura 3. Versão final do programa.

A figura 3 mostra a interface gráfica do simulador, que foi baseada na interface do projeto Formigas, desenvolvido na tese de doutorado do aluno Danilo Nogueira Costa, no grupo de Sistemas Distribuídos do ICMC-USP.

Não houve, entretanto, grande preocupação com relação à sua usabilidade, já que sua utilização seria limitada a testes do algoritmo e apresentações, sendo que a parte mais importante do projeto seria a otimização dos parâmetros através do Algoritmo Evolutivo proposto, que não possui interface gráfica.

3.4.2. A máquina de estados

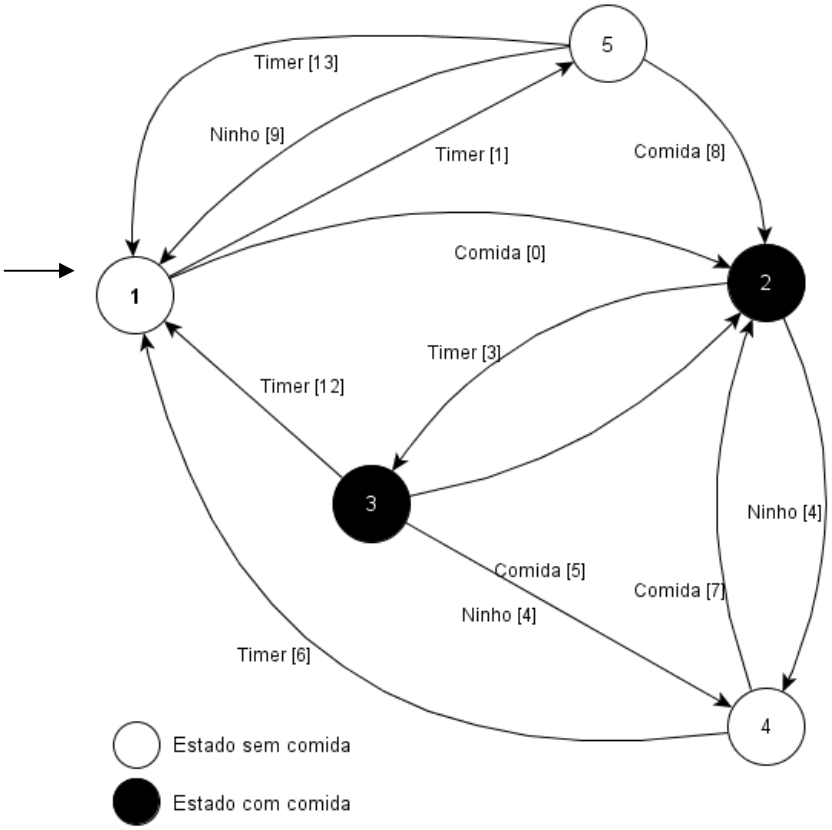

Figura 4. Estados básicos do sistema. Além dos estados e transições descritos, podem ser configuradas transições, baseadas em probabilidade, entre estados de mesma cor.

Cada formiga possui comportamento baseado em uma máquina de estados com cinco estados, descrita na Figura 4, sendo que o estado 1 é o inicial.

As funções lógicas que representam o comportamento das formigas foram pensadas da seguinte maneira:

Estado 1 (inicial): O ambiente deve ser explorado buscando-se encontrar comida. Espera-se que as configurações encontradas busquem fugir de feromônios, já que, desta forma, locais já explorados terão menos chances de serem explorados novamente.

Estado 2: Caso uma formiga encontre comida, seu estado é mudado para o estado 2. É esperado que, nesse estado, a formiga vire 180º e siga feromônios, de forma que provavelmente seguirá seu próprio feromônio até o local de onde veio, que possivelmente será o ninho. Se a quantidade de feromônio depositada for maior, o caminho até a comida poderá ser demarcado com mais força.

Estado 3: Se a formiga não encontrar o ninho após um determinado número de iterações, ela passará ao estado 3, no qual espera-se que o comportamento seja similar ao estado 2, porém depositando menos feromônio, já que ela provavelmente está seguindo a trilha errada.

Estado 4: Quando a formiga estava no estado 2 ou 3 (com comida) e chega ao estado 4, ela ganha um ponto no Algoritmo Evolutivo, pois ela levou comida até o ninho. Neste estado espera-se que a formiga tenha um comportamento similar ao do estado 3, porém buscando encontrar comida ao invés do ninho.

Estado 5: O estado 5 ocorre quando a formiga fica uma determinada quantidade de iterações no estado 1 e não encontra comida. Neste estado, espera-se que ela deposite pouco feromônio e siga-os, para tentar voltar ao ninho. Com este tipo de comportamento, espera-se que seja possível fazer com que as formigas se dirijam de volta do ninho e, caso alguma encontre comida, deixará um rastro mais forte que será seguido pelas outras formigas.

Apesar das idéias de configurações citadas acima, permitiu-se grande flexibilidade de configurações para os estados, de forma que é possível criar uma configuração onde um estado automaticamente mude para outro (configurando a probabilidade de mudar de estado como zero) ou definindo-se outras configurações de comportamento que não sejam as

esperadas.

3.4.3. Recursos do software

Configuração do ambiente

- **Ants**: Ao selecionar o botão “Set Nest”, a quantidade de formigas configurada no parâmetro “Ants” será inserida no ambiente na posição definida pelo usuário com o mouse.
- **Food**: Ao selecionar o botão “Set Food”, a quantidade de comida configurada no parâmetro “Food” será inserida no ambiente na posição definida pelo usuário com o mouse.
- **Draw**: Ao selecionar o botão “Fero”, a quantidade de feromônio configurada no parâmetro “Draw” será inserida no ambiente na posição definida pelo usuário com o mouse, para desenhar feromônios. O mesmo ocorre com o botão “Obstáculo”, porém o parâmetro “Draw” não tem efeito para o mesmo.

Visualização

- **Refreshes**: Intervalo entre interações em que a tela será redesenhada.
- **View sensor**: Quando esta opção é habilitada, os sensores de cada formiga são desenhados na tela.

Operações

- **Load**: Carrega um ambiente e configuração já salva.
- **Save**: Salva o ambiente e configurações atuais.
- **Reset**: Limpa o ambiente.
- **Run**: Inicia e pausa a simulação.
- **AG**: Inicia o modo visual e não distribuído do Algoritmo Evolutivo, que mostrará graficamente o resultado final da avaliação de cada indivíduo da população.

Configuração das formigas

- **Pheromones**
    - **Evaporate {X} each {Y} interactions:** quantidade X de feromônios a serem

evaporados a cada Y interações.

- Saturation: Nenhum ponto do ambiente poderá ter mais feromônios do que especificado por este parâmetro. Supõe-se que, para um valor de saturação baixo, poderá haver uma tendência maior das formigas saírem da trilha atual, migrando para uma outra.

Turn prob: Probabilidade de virar cada ângulo especificado, quando não influenciada por feromônios ou outros objetos. A probabilidade é calculada pela fórmula

p = (**Turn prob**) / 1000

Configurações de comportamento 1, 2 e 3

Os parâmetros abaixo configuram cada uma dos três comportamentos possíveis para uma formiga em cada estado. Existe um parâmetro em cada estado que irá selecionar qual dos três comportamentos será utilizado.

- Follow F1: Probabilidade de virar para o lado com maior concentração de feromônio, desde que o limiar seja respeitado. A probabilidade é calculada pela fórmula

p = (**Follow F1**) / 1000

- Threshold: Limite inferior e superior para que a concentração de feromônio de um lado da formiga seja considerada significativamente maior que a de outro.

- Against F1: Probabilidade de virar para o lado com menor concentração de feromônio, desde que o limiar seja respeitado. A probabilidade é calculada pela fórmula

p = (**Against F1**) / 1000

- Hard follow: Se habilitada, a formiga irá sempre para o lado com maior (se estiver seguindo) concentração de feromônio ou menor (se estiver fugindo), desde que a probabilidade de seguir ocorra e respeite o limiar. Se desabilitada, mesmo que isso ocorra, a escolha será também ponderada pela diferença de concentração. Por exemplo: neste último caso, se o lado esquerdo recebe 1000 na contagem de feromônios, o lado direito 3000 e a formiga decide seguir feromônio, ela virará para o lado esquerdo com 25% de chance (1000/(1000+4000) = 0,25) e para o lado direito com 75%.

- Relative Thr: Se habilitada, o limiar será calculado com a fórmula 1000 * (lado1 / lado2). Se desabilitada, a fórmula será (lado1 – lado2). O lado1 será sempre o lado com maior concentração.

Os seguintes parâmetros são comuns aos Estados 1, 2, 3, 4 e 5:

- **Deposit F1**: Quantidade de feromônio que a formiga deposita ao andar.
- **Time limit**: Quantidade máxima de iterações que a formiga permanecerá no estado. Ao atingir o limite, ela mudará para o estado indicado no diagrama de estados na Figura 2.
- **Sensor radius**: Raio do sensor da formiga.
- **Sensor position**: Posição do sensor no eixo do movimento, sendo que o valor 0 significa que o centro do sensor estará posicionado a uma distância igual ao **Sensor radius** para trás do centro da formiga, como pode ser visto na Figura 5. Com o valor do **Sensor radius** configurado, o centro do sensor coincidirá com o centro da formiga, como na Figura 5. O valor do **Sensor position** pode chegar até 2***Sensor radius**, que posicionará o centro do sensor à distância **Sensor radius** à frente do centro da formiga.

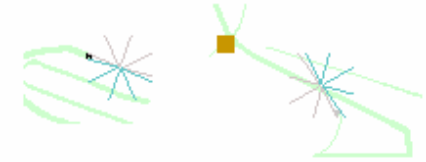

Figura 5. Sensor na frente da formiga e no centro, respectivamente.

- **Behavior config**: Define qual das três configurações de comportamento que será utilizada no estado.
- **Prob change to state**: Probabilidade (1 em X) de mudar para o estado especificado.

Turn angle: Ângulo que a formiga virará ao passar pela transição X (especificada entre colchetes no diagrama de estados da Figura 4). Supõe-se que, por exemplo, ao passar de um estado sem comida para um estado com comida, a formiga deva virar 180º para tentar voltar pela mesma trilha que veio. Pode ser interessante também que ela vire, por exemplo, 135º, o que permitiria que ela fugisse de sua trilha com maior probabilidade, desde que encontrasse outra trilha mais forte ao redor.

Radius decay: Decaimento (entre 0 e 1) da quantidade de feromônio somada em função da distância do centro da formiga. Para distância N, soma-se **Radius decay** ^{N} * (quantidade de

feromônio). O objetivo deste parâmetro é permitir que a formiga possa ponderar o quanto deve se basear na trilha em que está no momento (possivelmente criada por ela mesmo) em relação a trilhas próximas.

Para se encontrar uma configuração ideal de parâmetros, seria necessário executar-se testes com todas as combinações de parâmetros possíveis. Se feito desta forma, o espaço de busca do Algoritmo Evolutivo seria muito grande, de forma que os parâmetros foram discretizados ponderando-se uma boa relação entre o tamanho do espaço de busca final com a precisão necessária. Os seguintes valores podem ser configurados, pelo Algoritmo Evolutivo, para os respectivos parâmetros:

- Pheromones

- Evaporate [X] each [Y] interactions: [X] = 0, 1, 2, 4, 8; [Y] = 4, 8, 16, 32, 64

- Saturation: 128, 512, 4096, 16384, 65536, 2000000000

Turn prob: 0, 2, 5, 10, 20, 40, 80, 160, 240

Configuração de comportamento 1, 2 e 3:

- Follow F1: 0, 2, 5, 10, 20, 40, 80, 160, 240, 320, 500, 680, 840, 920, 960, 980, 990, 995, 998, 1000;

- Limiar: 0, 10, 25, 50, 100, 150, 200, 275, 350, 400, 800, 1600, 3200, 6400, 9600, 12800, 25600, 2000000000

- Against F1: 0, 2, 5, 10, 20, 40, 80, 160, 240, 320, 500, 680, 840, 920, 960, 980, 990, 995, 998, 1000 (limitado pela expressão FollowF1 + AgainstF1 <= 1000)

- Hard follow: 0, 1

- Relative Thr: 0, 1

Estados 1, 2, 3, 4 e 5:

- Deposit F1: 0, 1, 2, 4, 8, 16

- Time limit: 0, 10, 25, 50, 100, 150, 200, 275, 350, 400, 800, 1600, 3200, 6400

- Sensor radius: 0, 2, 16

- Sensor position: 14, 16, 18, 28, 30, 32

- Behavior config: 1, 2, 3

- Prob change to state: 0, 10, 25, 50, 100, 150, 200, 275, 350, 400, 800, 1600, 3200, 6400, 9600, 12800, 25600, 2000000000

Turn angle: 0, 2, 4, 6, 18, 30, 32, 34, 36

Radiusdecay: 0,65, 0,80, 0,85, 0,90, 0,95, 0,96, 0,97, 0,98, 0,985, 0,99, 0,995, 1,00

4. Experimentos

Os experimentos foram desenvolvidos buscando-se evitar alguns comportamentos indesejáveis que Algoritmos Evolutivos podem possuir, como a produção de soluções muito específicas. Isso ocorre quando é encontrada uma solução com nota de avaliação muito boa, porém indesejável, para o experimento. Por exemplo, o Algoritmo Evolutivo poderia encontrar, se existisse, uma configuração onde as formigas fizessem sempre um mesmo caminho, que coincidisse com a localização da comida no experimento. Desta forma, caso o ninho fosse deslocado, a comida não seria mais encontrada com eficiência similar. Para tentar evitar problemas como esse, alguns experimentos foram executados com dois ambientes distintos e a pontuação final é a média harmônica dos mesmos, já que se prefere uma configuração em que a função de avaliação de ambos os ambientes sejam similares a uma configuração com muita variação entre os ambientes, mesmo que esta tenha média aritmética maior.

Além disso, todos os experimentos foram efetuados com formigas saindo do formigueiro na mesma posição inicial, a fim de que o ângulo de saída delas não influenciasse o experimento de maneira incorreta (por exemplo, poderia ocorrer de três formigas saírem direto em direção à comida em um experimento e nenhuma em outro).

Por trabalhar com probabilidades que possam influenciar a nota de cada ambiente, são feitos sempre três testes, buscando-se minimizar o efeito aleatório introduzido. A pontuação de cada ambiente é calculado pela média ponderada dos valores dos três

experimentos como 1*menor + 2*médio + 1*maior (sendo que menor e maior são, respectivamente, a menor e maior pontuação obtida nas três execuções com sementes de aleatoriedade distintas; e médio é a pontuação obtido na outra execução).

Em todos os experimentos são executadas 5000 iterações do algoritmo, com ninho de 25 formigas, de forma que a função de avaliação é definida como “quantidade de comida levada ao ninho, por 25 formigas, em 5000 iterações”. A comida é ilimitada.

4.1. Experimento 1

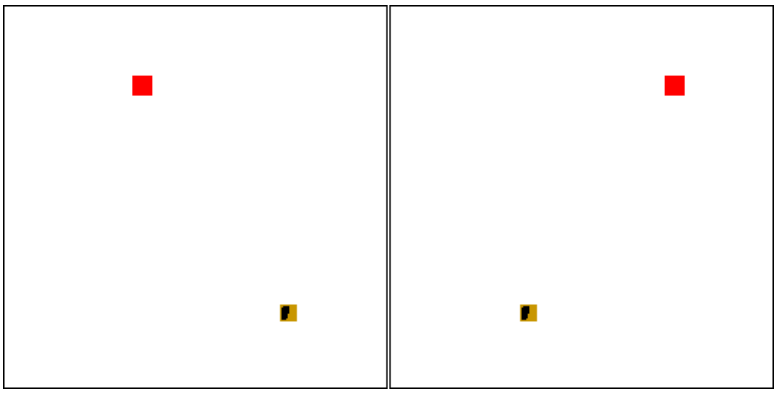

Figura 6. Ambientes do experimento 1, onde o ninho aparece em bege e as fontes de alimento em vermelho.

O Experimento 1 possui dois ambientes (ver Figura 6). No primeiro, há um ninho na posição (185, 200), sendo que a matriz que define o ambiente possui 250 por 250 pontos e o ponto (0, 0) é definido no canto superior esquerdo, e um foco de comida na posição (90, 52). No segundo, o ninho está na posição (90, 200) e o foco de comida está na posição (185, 52).

Espera-se que seja encontrada uma configuração que otimize a busca de comida no único foco disponível, possivelmente fazendo uma trilha reta, ou próxima de reta, do formigueiro até o foco de comida.

4.2. Experimento 2

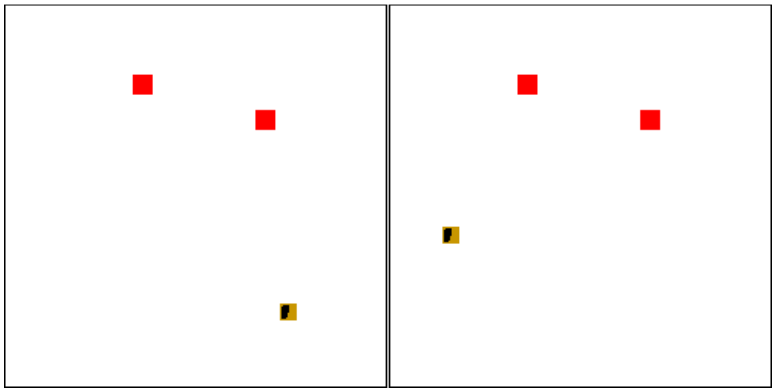

Figura 7. Ambientes do experimento 2.

No experimento 2, realizado em dois ambientes (ver Figura 7), há dois focos de comida com distâncias diferentes em relação ao ninho.

No primeiro ambiente, o ninho é configurado na posição (40, 150) e os focos de comida são configurados nas posições (90, 52) e (170, 75). No segundo ambiente, o ninho é configurado na posição (185, 200) e a posição dos focos de comida não é alterada.

Espera-se que seja encontrada uma configuração com tendência a buscar comida no foco de comida mais próximo do ninho, fazendo uma trilha otimizada entre este e o ninho.

4.3. Experimento 3

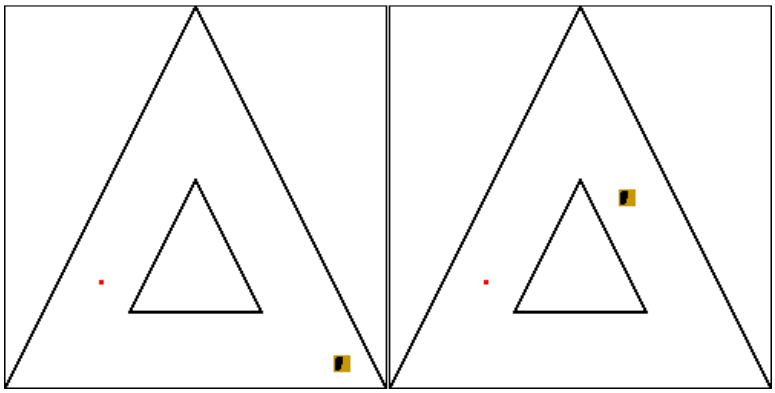

Figura 8. Ambiente do experimento 3.

O experimento 3 também tem como objetivo a busca do menor caminho entre o ninho e a comida, porém a locomoção das formigas é restringida pelos ambientes (ver Figura 8), que possuem obstáculos que não podem ser ultrapassados.

No primeiro ambiente, o ninho se localiza na posição (220, 233) e a comida na posição (63, 180). No segundo ambiente, o ninho é movido para a posição (155, 125), e a comida é mantida na mesma posição.

4.4. Experimento 4

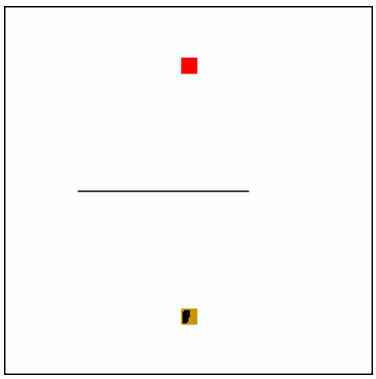

Figura 9. Ambiente do experimento 4.

No experimento 4 (ver Figura 9) foi configurado um obstáculo intransponível exatamente entre o ninho (posição (125, 210)) e a comida (posição (125, 40)). O obstáculo está ligeiramente deslocado para a esquerda, de forma que o menor caminho do ninho à comida será pela direita.

4.5. Experimento 5

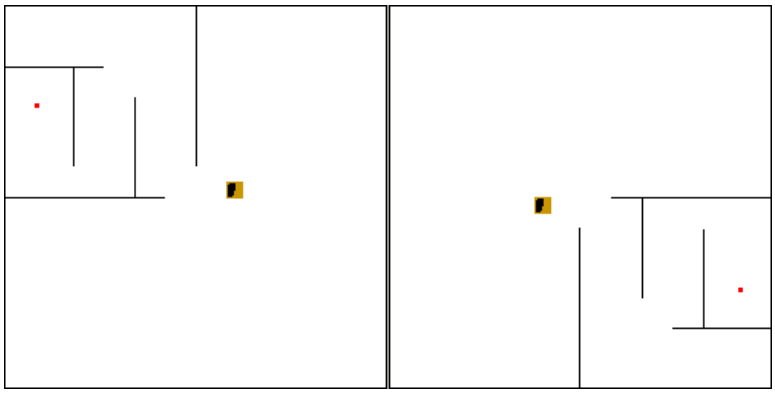

Figura 10. Ambiente do experimento 5.

O experimento 5 é baseado em um formato de labirinto simples Figura 10, para o qual espera-se que algumas formigas consigam encontrar o foco de comida e voltar para o ninho.

O ninho está localizado na posição (150, 120) e a comida na posição (25, 60).

5. Resultados e discussão

As configurações da Tabela I do Apêndice I foram obtidas pelo Algoritmo Evolutivo em um cluster de 14 nós com processador Intel(R) Pentium(R) 4 CPU 3.40GHz dual e sistema operacional Linux. O critério de parada do processo evolutivo foi o seguinte: se o valor da pontuação não melhorar por pelo menos 100 gerações, o Algoritmo Evolutivo é encerrado.

5.1. Experimento 1

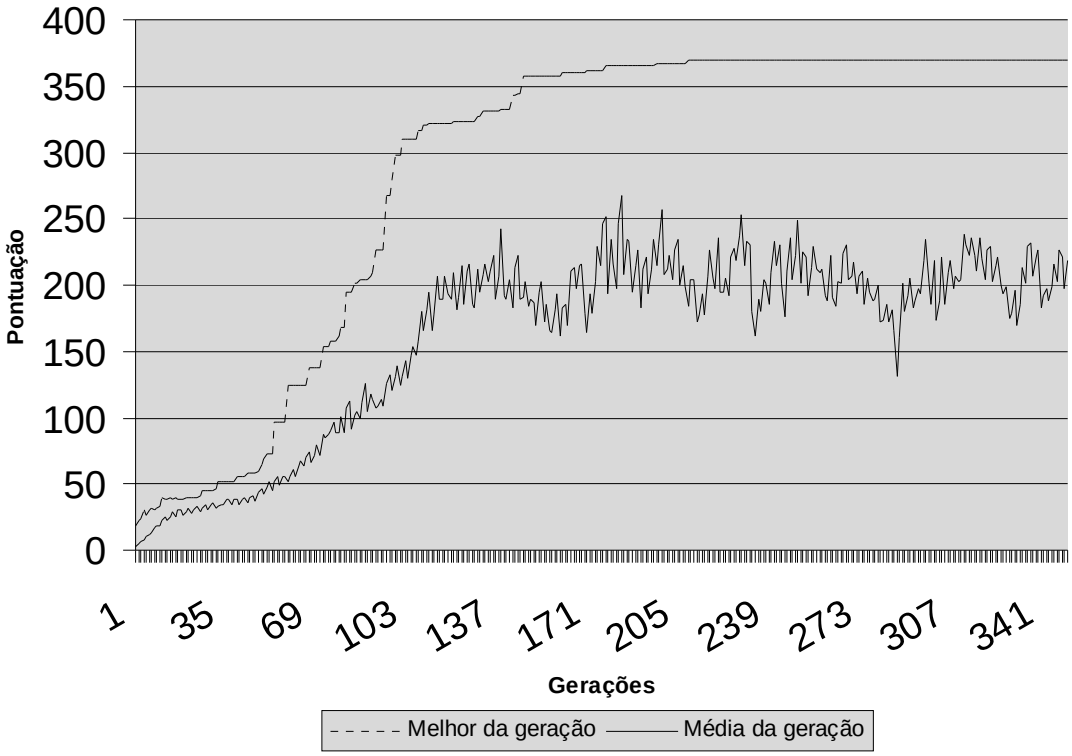

Figura 11. Pontuações obtidas pelo Algoritmo Evolutivo em 350 gerações.

Para este experimento, o Algoritmo Evolutivo teve uma convergência rápida, em aproximadamente 150 gerações, sendo que gerações posteriores tiveram pouca variação da pontuação máxima que havia sido obtida, como pode ser visto na Figura 11.

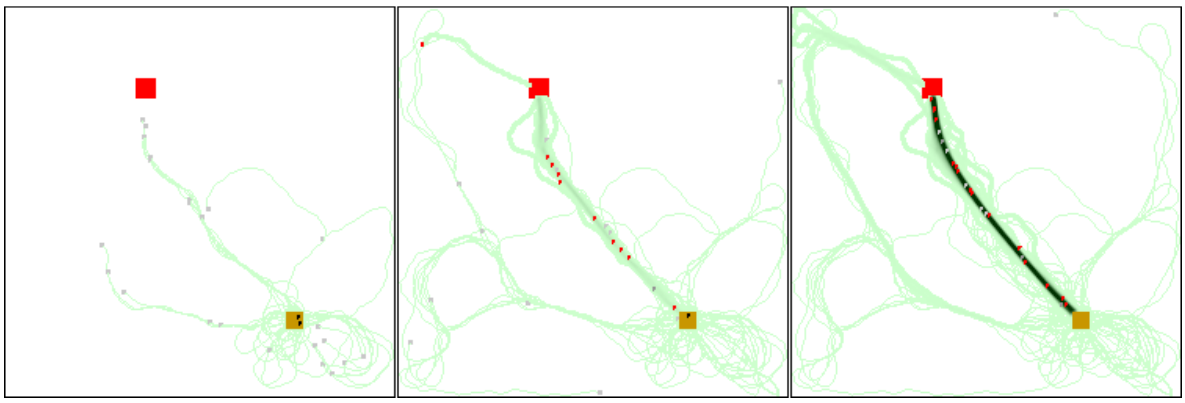

Figura 12. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

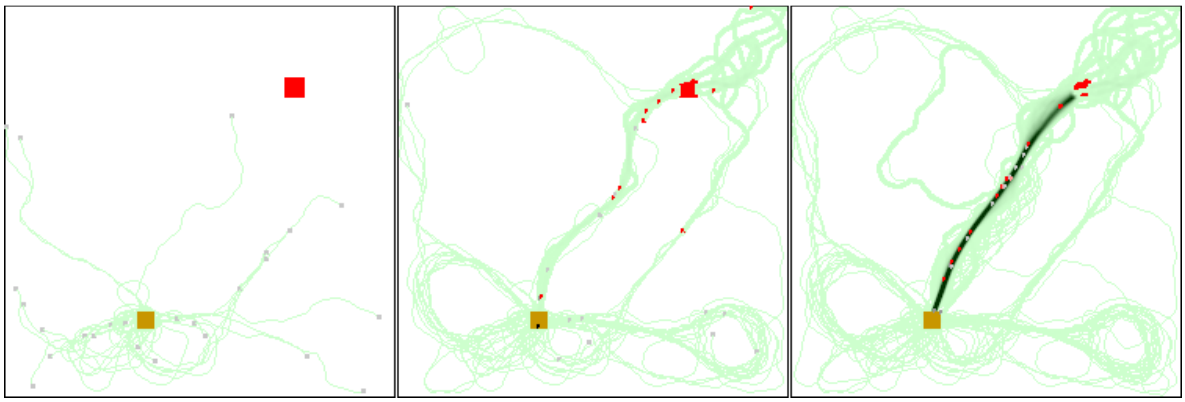

Figura 13. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

As Figuras 12 e 13 mostram o comportamento do formigueiro em alguns estágios (150, 500 e 3000 interações) para os dois ambientes de testes propostos.

Pode-se verificar pela Tabela I do Apêndice I que a configuração obtida trabalha com somente quatro estados, dos cinco disponíveis, já que o estado 1 foi configurado para imediatamente transferir para o estado 5 (através do parâmetro "Prob change to state 5" configurado como "1 em 1").

No estado 5, que passa então a ser o inicial, as formigas depositam pouco feromônio, possuem o sensor na frente, e são configuradas com comportamento principalmente de seguir feromônio, o que não era esperado, devido à necessidade de explorar o ambiente para encontrar comida. É provável que tenha sido encontrado, pelo Algoritmo Evolutivo, uma configuração específica, já que foram executados somente três

testes para cada ambiente, que podem ter coincidido de, na média, encontrar a fonte de comida rapidamente.

Quando encontra comida, a formiga muda para o estado 2, que deposita mais feromônios, como era esperado, reforçando a trilha. Este estado, entretanto, não possui configuração para seguir ou fugir de feromônio, já que o limiar configurado nunca será atingido (o mínimo foi configurado como sendo maior que o máximo), desta forma, a formiga é guiada somente pelas probabilidades de "turn". Supõe-se que essa configuração teve bom resultado, pois a formiga foi configurada para virar 180º quando encontrar comida estando no estado 5.

Ao retornar para o ninho após encontrar comida, a formiga passa ao estado 4, que foi configurado para seguir e depositar feromônio.

É interessante notar que a evaporação de feromônio não foi habilitada, o que pode ser explicado pela invariância do ambiente e único foco de comida. Se a comida fosse limitada e pudesse ser necessário buscar outro foco de comida, provavelmente uma configuração sem evaporação de feromônio seria inferior.

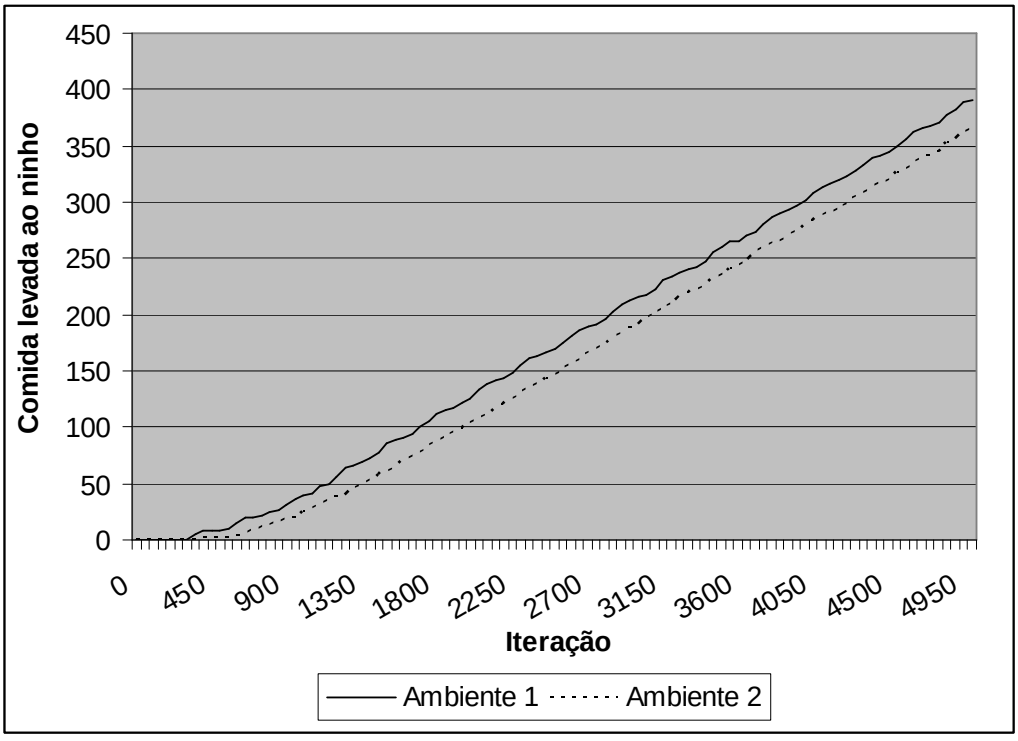

Figura 14. Total de comida levada ao ninho por iteração.

A Figura 14 representa a quantidade total de comida levada ao ninho em função do número de iterações. Pode-se verificar que as formigas demoram aproximadamente 300 iterações para levarem comida inicialmente e, após 500 iterações, estabilizam um padrão de forrageamento.

5.2. Experimento 2

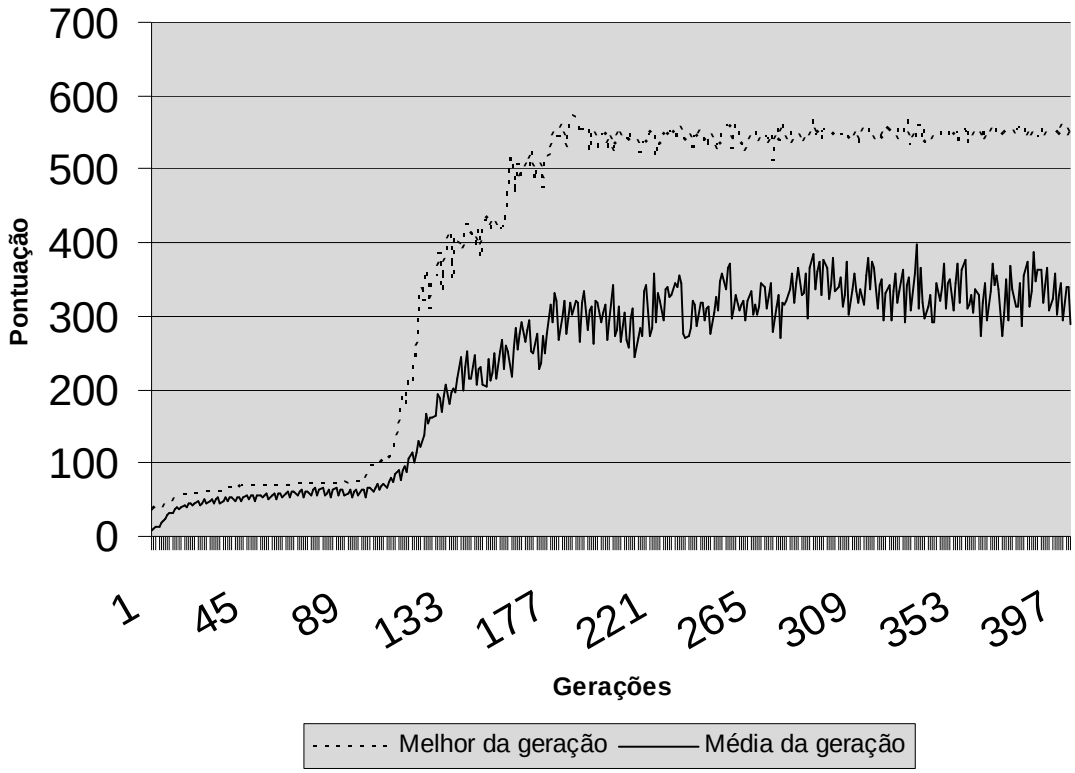

Figura 15. Pontuações obtidas pelo Algoritmo Evolutivo em 400 gerações.

Neste experimento, o Algoritmo Evolutivo convergiu em aproximadamente 200 gerações, sendo que gerações posteriores tiveram pouca variação da pontuação máxima que havia sido obtida, como pode ser visto na Figura 15.

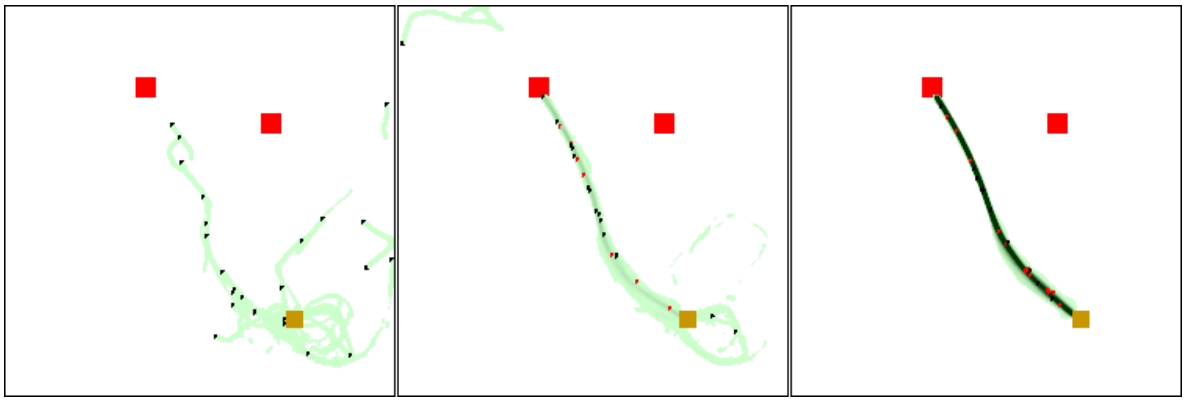

Figura 16. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

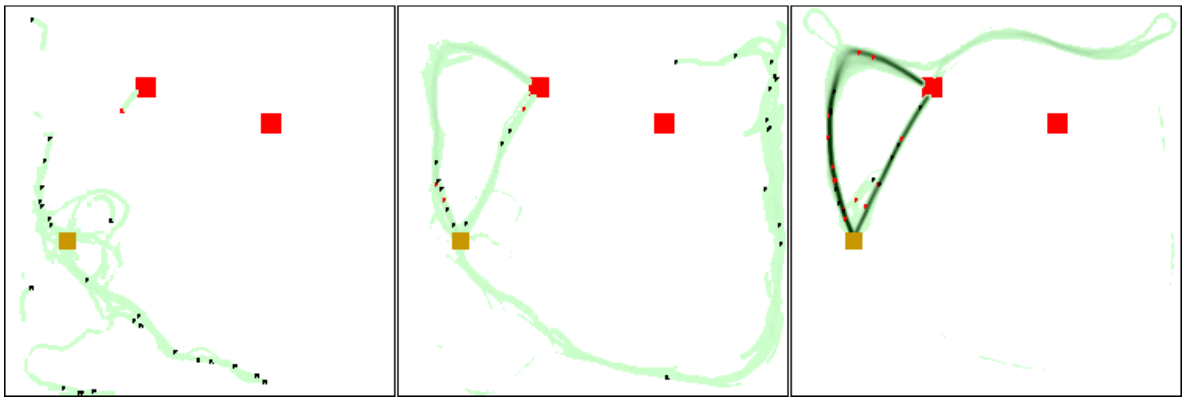

Figura 17. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

As Figuras 16 e 17 mostram o comportamento do formigueiro em alguns estágios (150, 500 e 3000 interações) para os dois ambientes de testes propostos.

Para a configuração deste ambiente, nota-se que a evaporação de feromônio foi ativada, porém em quantidade relativamente baixa (em média 1 a cada 32 iterações).

O estado 1 deposita menos feromônio que o estado 2, como era esperado, porém também segue feromônios, ao invés de fugir dos mesmos para possibilitar a exploração do ambiente. Isto pode ser explicado também como no experimento 1, além do limite do tamanho do ambiente. Com um ambiente maior, a dificuldade de se encontrar comida aleatoriamente seria maior, portanto essa configuração poderia ser mais interessantes para ambientes diferentes dos que foram testados, nos quais normalmente a comida é encontrada de qualquer maneira, o que também foi facilitado por, neste experimento, haver dois focos de comida.

Pode-se notar um padrão de “fila” das formigas, por conta do comportamento de seguir feromônio. Este padrão é interessante, pois, caso a primeira formiga encontre alimento inicialmente, será imediatamente formada uma trilha do ninho ao formigueiro. Uma desvantagem deste comportamento é exatamente a que pode ser visualizada no resultado dos experimentos: quando caminhos subótimos são encontrados, normalmente eles são mantidos, ao invés de haver uma convergência para um caminho mais otimizado.

O estado 4 foi configurado para ser transferido para o estado 1, de forma que a configuração feita utiliza-se basicamente de apenas dois estados (1 e 2).

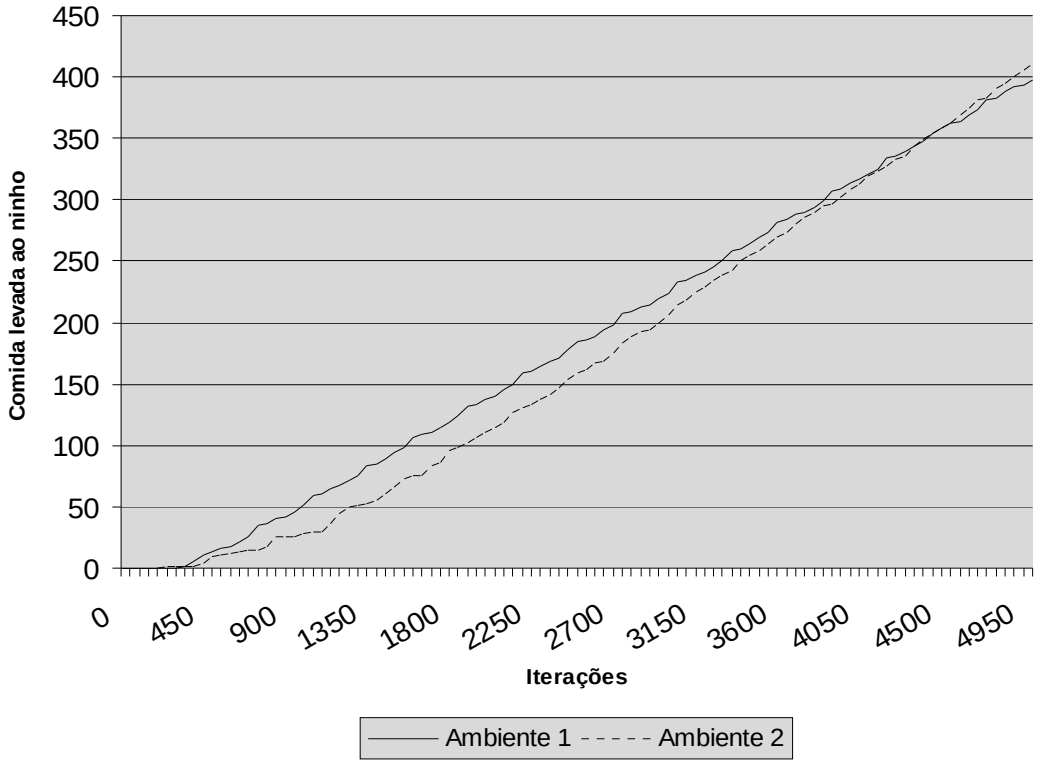

Figura 18. Total de comida levada ao ninho por iteração.

A Figura 18 representa a quantidade total de comida levada ao ninho em função do número de iterações. Pode-se verificar que as formigas demoram aproximadamente 450 iterações para levarem comida inicialmente e, após 600 iterações, estabilizam um padrão de forrageamento.

5.3. Experimento 3

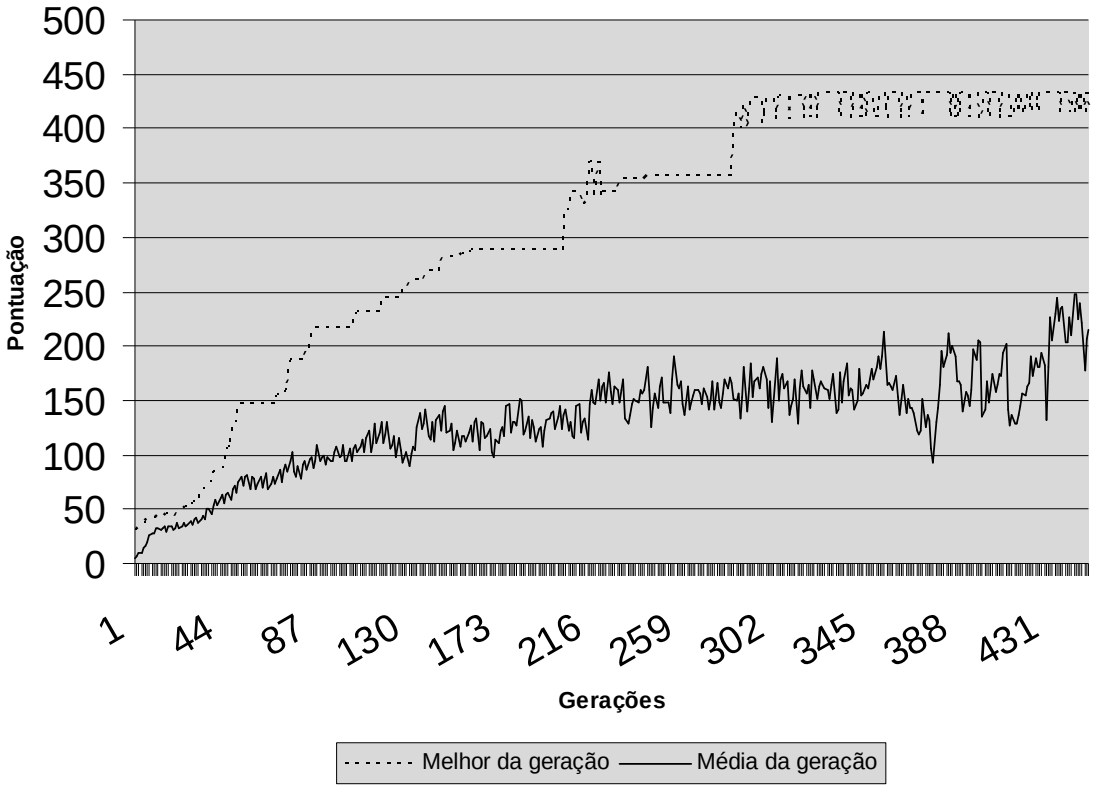

Figura 19. Pontuações obtidas pelo Algoritmo Evolutivo em 450 gerações.

A convergência do Algoritmo Evolutivo ocorreu em aproximadamente 300 gerações para este algoritmo, sendo que gerações posteriores tiveram pouca variação da pontuação máxima que havia sido obtida, como pode ser visto na Figura 19.

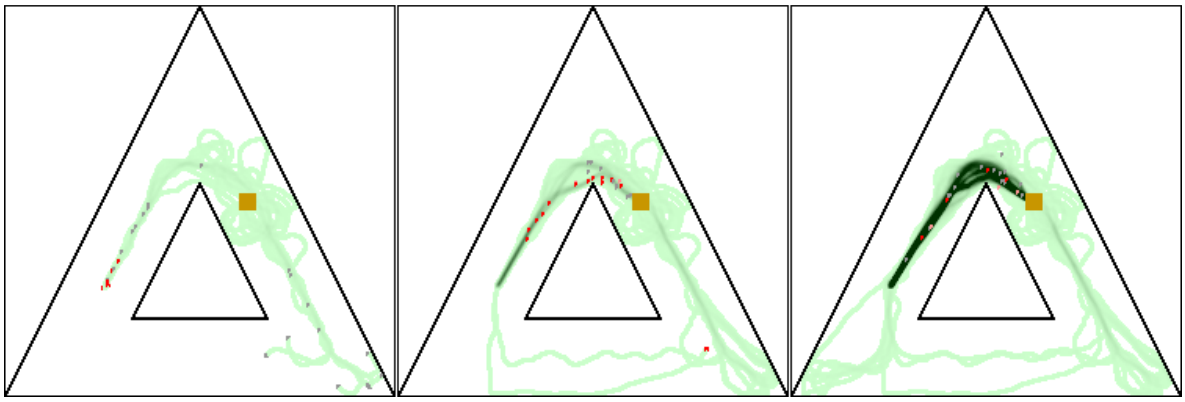

Figura 20. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

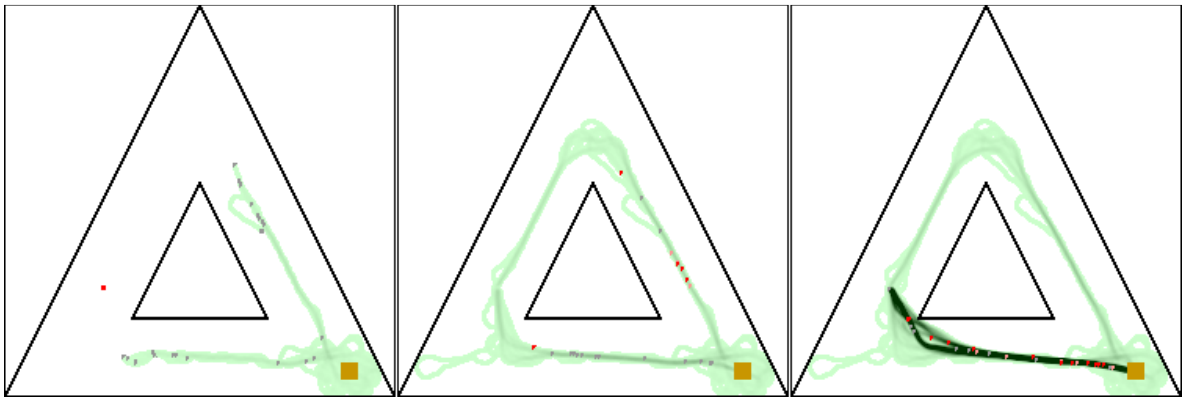

Figura 21. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

As Figuras 20 e 21 mostram o comportamento do formigueiro em alguns estágios (150, 500 e 3000 interações) para os dois ambientes de testes propostos. Este experimento é interessante pois há somente duas trilhas do ninho à comida: uma longa e uma curta. O fato de serem utilizados dois ambientes em condições distintas fez com que o Algoritmo Evolutivo encontrasse uma solução muito interessante, para a qual, nos experimentos feitos com a mesma, a melhor trilha era seguida na grande maioria dos casos.

Pode-se notar também, no gráfico da quantidade de comida levada ao ninho por interação, que claramente no ambiente 1 mais comida é levada que no ambiente 2. Isto ocorre pois, no ambiente 1, a distância entre o ninho e a comida é menor.

A configuração encontrada não utiliza evaporação. Como há basicamente somente duas trilhas do ninho à comida, a configuração do estado 1 para seguir feromônios,

novamente formando um padrão de fila, é interessante pois é esperado que algumas formigas iniciem em uma das trilhas disponíveis e outras na outra, de forma que o ambiente será bem explorado neste caso.

Novamente o estado 4 não foi utilizado, de forma que a configuração encontrada utiliza somente dois estados (1 e 2).

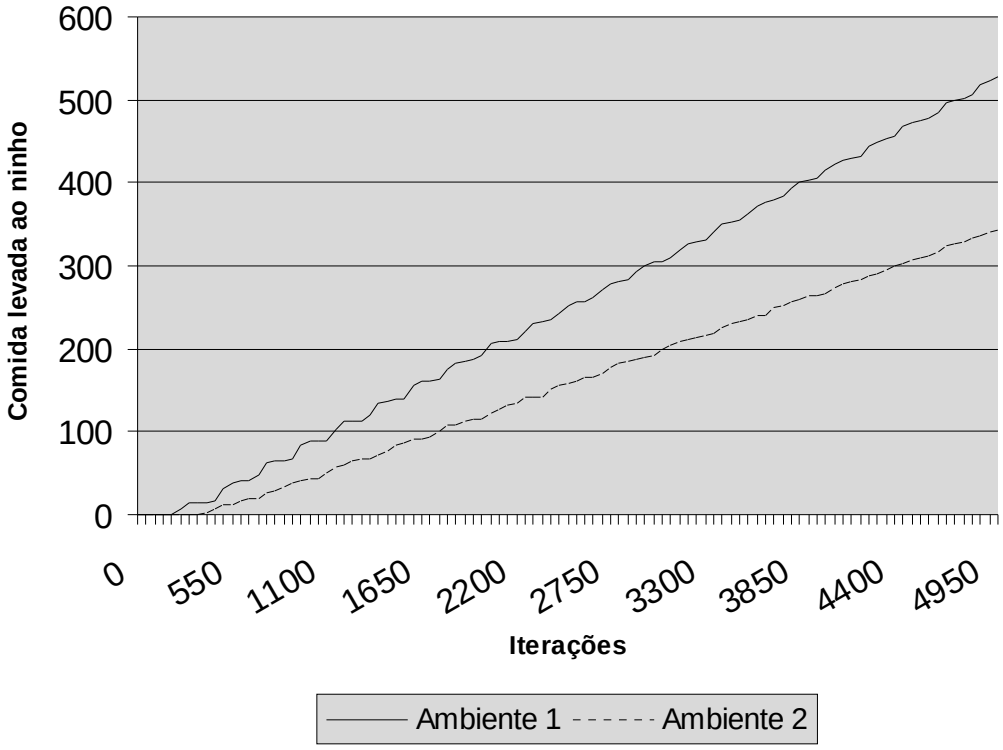

Figura 22. Total de comida levada ao ninho por iteração.

A Figura 22 representa a quantidade total de comida levada ao ninho em função do número de iterações. Pode-se verificar que as formigas demoram aproximadamente 300 iterações para levarem comida inicialmente e, após 500 iterações, estabilizam um padrão de forrageamento.

5.4. Experimento 4

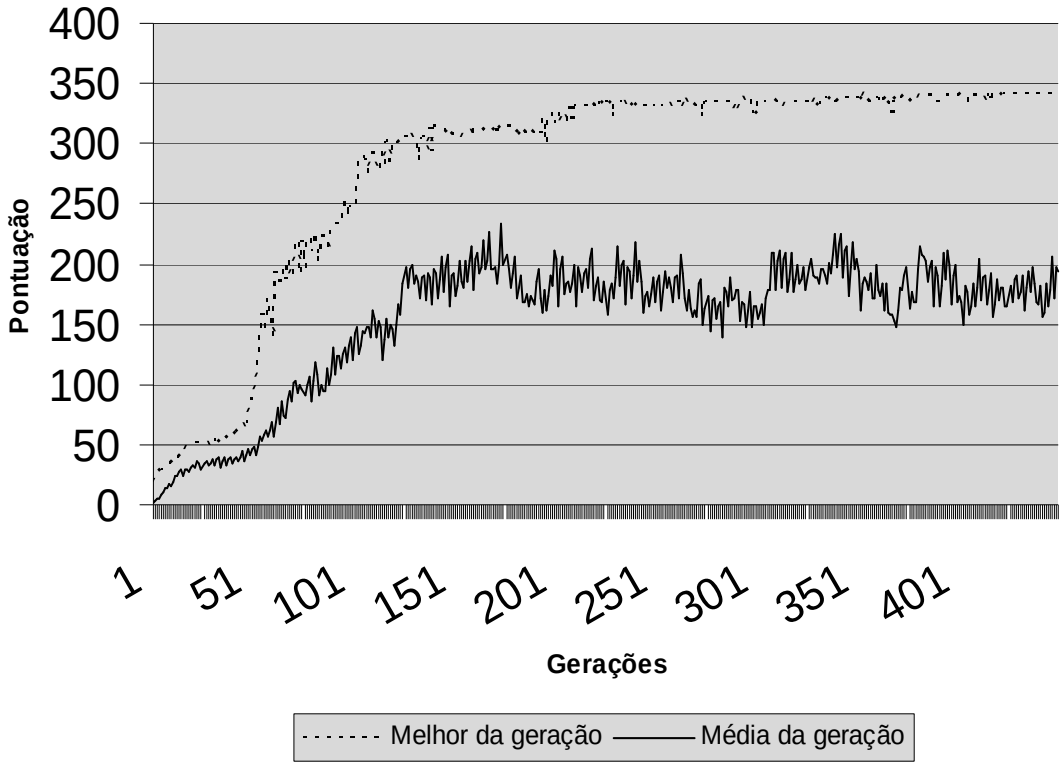

Figura 23. Pontuações obtidas pelo Algoritmo Evolutivo em 450 gerações.

A convergência do Algoritmo Evolutivo ocorreu em aproximadamente 150 gerações para este algoritmo, sendo que gerações posteriores tiveram pouca variação da pontuação máxima que havia sido obtida, como pode ser visto na Figura 23.

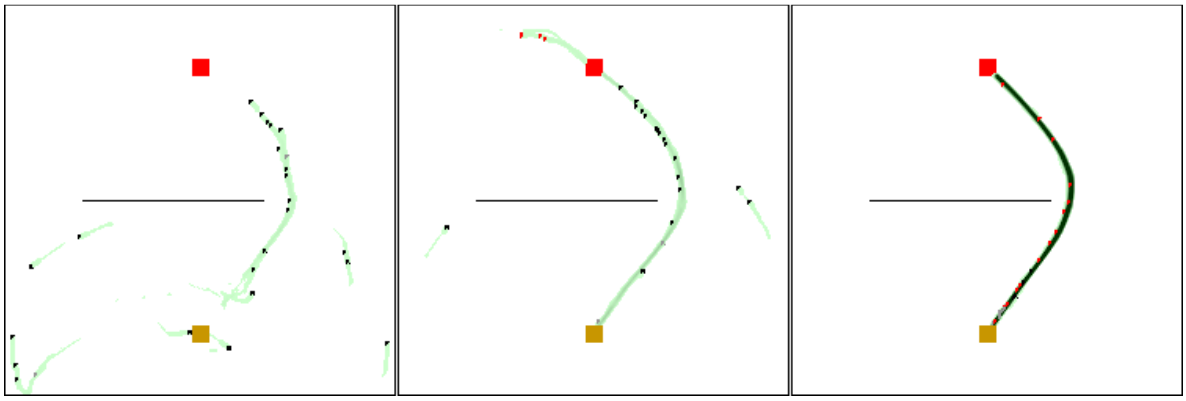

Figura 24. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

A Figura 24 mostra o comportamento do formigueiro em alguns estágios (150, 500 e 3000 interações) para o ambiente de testes propostos. Este experimento utiliza muita evaporação (em média 1 a cada 2 iterações) e faz poucas curvas (menos de 8% das iterações). Com exceção do estado 5, que dificilmente é chamado pois o timer do estado 1 é alto, bem como as probabilidades de transição para o mesmo, todos os estados depositam a mesma quantidade de feromônio. O estado 1 segue feromônios, mostrando novamente o padrão de fila, que provavelmente foi escolhido neste experimento por haver somente um ambiente de testes, de forma que a comida pode ter sido encontrada, em média, eficientemente nos somente três testes que são feitos.

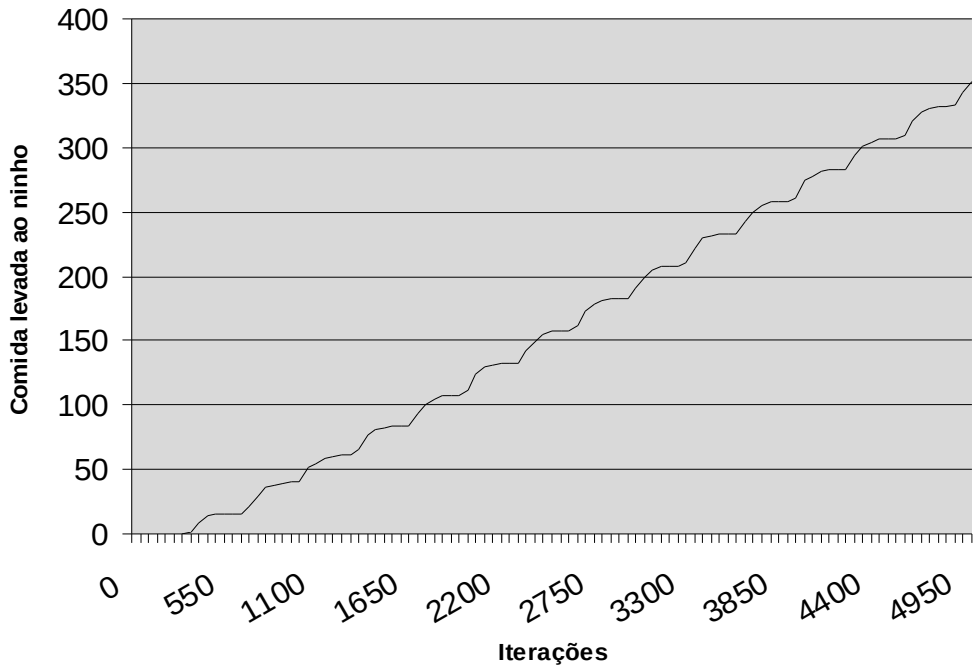

Figura 25. Total de comida levada ao ninho por iteração.

A Figura 25 representa a quantidade total de comida levada ao ninho em função do número de iterações. Pode-se verificar que as formigas demoram aproximadamente 500 iterações para levarem comida inicialmente e, após 700 iterações, estabilizam um padrão de forrageamento. É interessante notar o padrão de “escada” na quantidade de comida levada ao ninho por iteração, gerado pelo padrão de fila das formigas, que tendem a seguir a formiga da frente.

5.5. Experimento 5

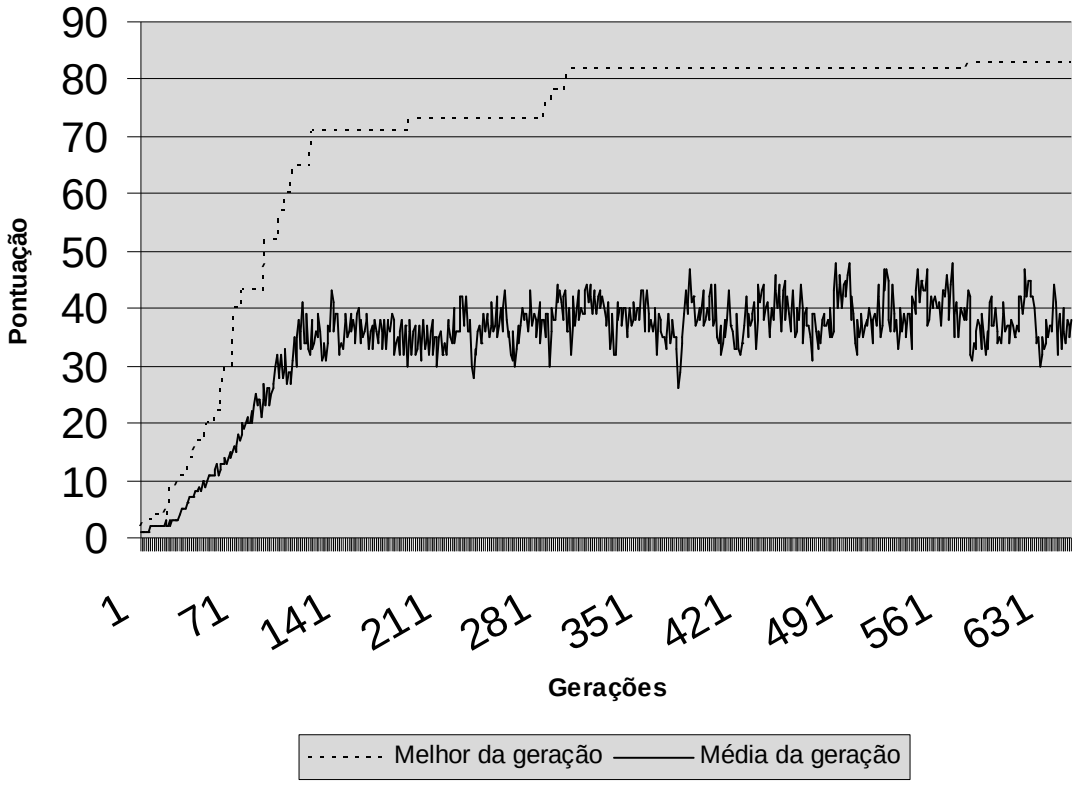

Figura 26. Pontuações obtidas pelo Algoritmo Evolutivo em 600 gerações.

A convergência do Algoritmo Evolutivo ocorreu em aproximadamente 350 gerações para este algoritmo, sendo que gerações posteriores tiveram pouca variação da pontuação máxima que havia sido obtida, como pode ser visto na Figura 25.

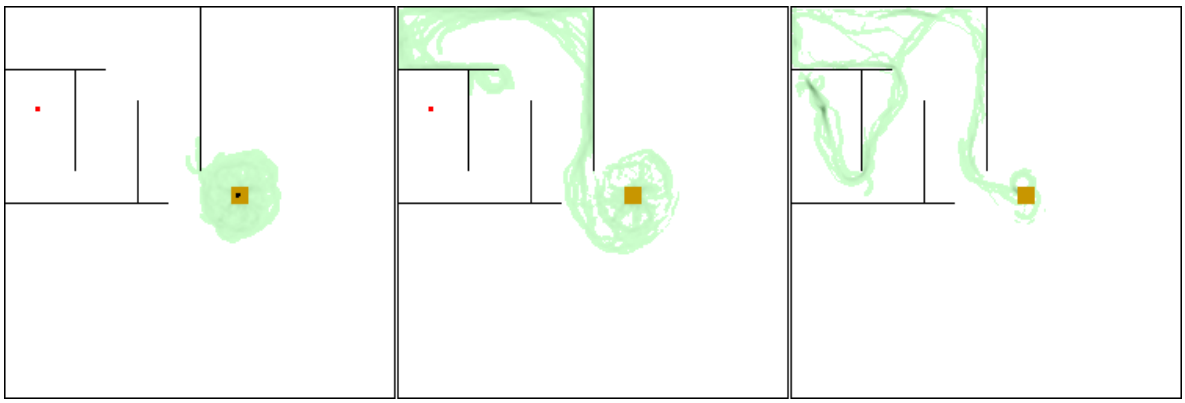

Figura 27. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

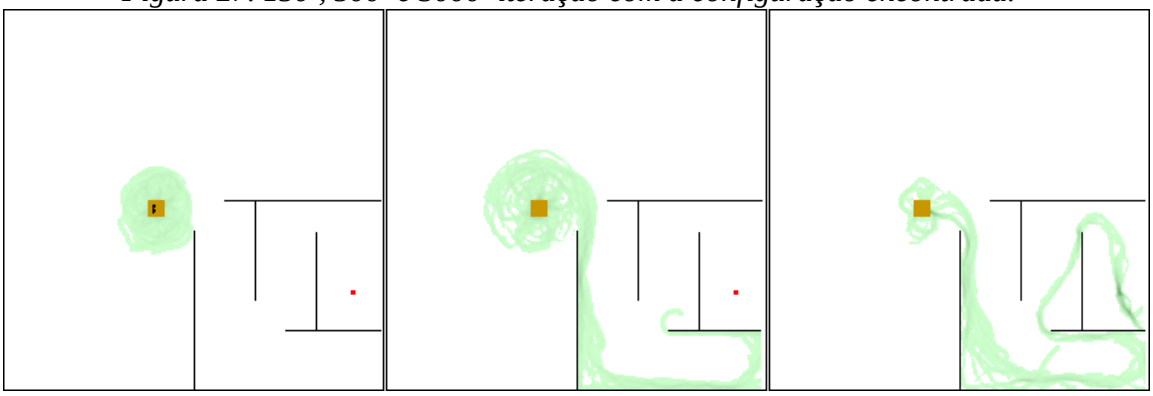

Figura 28. 150ª, 500ª e 3000ª iteração com a configuração encontrada.

As Figuras 27 e 28 mostram o comportamento do formigueiro em alguns estágios (150, 500 e 3000 interações) para os dois ambientes de testes propostos.

A configuração encontrada pelo Algoritmo Evolutivo é interessante, pois ele conseguiu encontrar uma configuração que fizesse com que as formigas andassem em torno do ninho, no sentido horário, até que o obstáculo fosse encontrado e, assim, as formigas pudessem explorar essa área buscando a comida.

É interessante notar que o resultado desse experimento foi claramente voltado especificamente à solução da configuração do ambiente proposto no experimento, pois, por girar no sentido horário, a configuração é ruim caso o labirinto esteja em algum outro canto

do ambiente. Essa é, como descrito anteriormente, uma desvantagem dos Algoritmos Evolutivos, que pode ser resolvida com mais casos de testes e ambientes, buscando eliminar o viés das soluções encontradas.

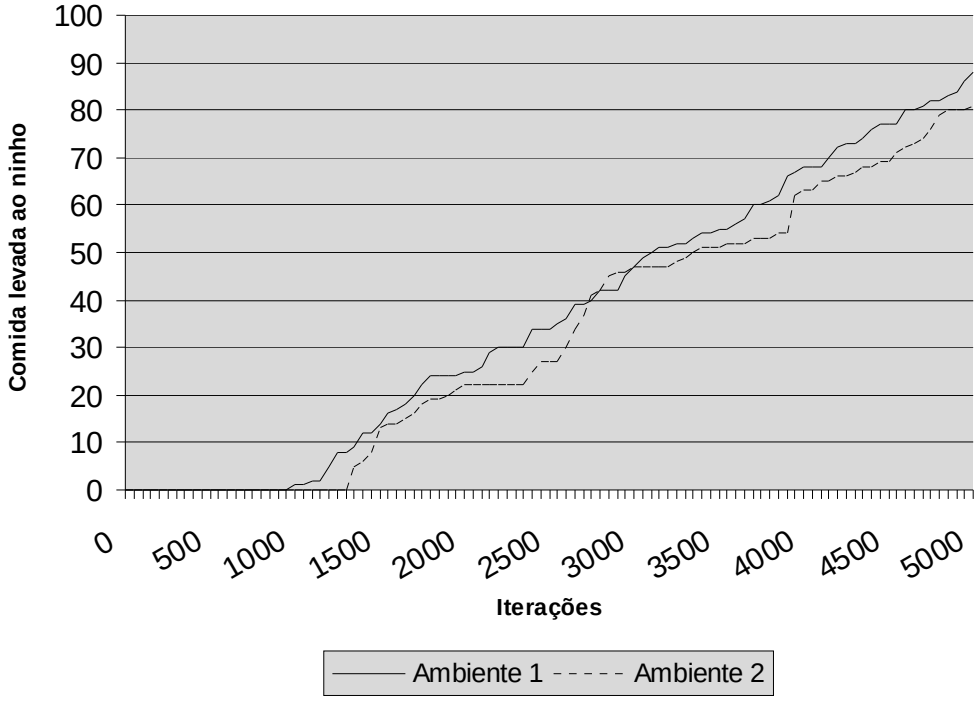

Figura 29. Total de comida levada ao ninho por iteração.

A Figura 29 representa a quantidade total de comida levada ao ninho em função do número de iterações. Pode-se verificar que as formigas demoram mais de 1000 iterações para levarem comida inicialmente, exatamente pelo padrão da solução encontrada, que fica girando em torno do ninho até encontrar o primeiro obstáculo. A solução não estabilizou um padrão de forrageamento como nos outros experimentos, mas o este ocorreu de forma eficiente.

6. Conclusão

A partir dos resultados obtidos, conclui-se que os parâmetros disponíveis para configuração são suficientes para adaptação a diversos ambientes, gerando bons resultados, como os obtidos nos experimentos, nos quais sempre foi possível encontrar caminhos

otimizados.

Pôde-se notar, entretanto, que, em alguns exemplos, apesar de terem sido adaptados com três *seeds* aleatórias distintas, o resultado obtido quando testado com outras *seeds* ou pequenas alterações pode ser até 30% inferior, o que indica que o Algoritmo Evolutivo em questão tende a gerar soluções um tanto específicas para as configurações de ambiente utilizadas para o processo evolutivo. Sugere-se que, futuramente, em um cluster com maior capacidade de processamento, ou com mais tempo disponível, sejam feitos experimentos mais precisos, com uma quantidade maior de ambientes e *seeds*, buscando-se obter uma configuração estável quando defrontada com pequenas modificações.

Com este trabalho, aprendeu-se não somente o suficiente para desenvolvimento do ambiente de simulação, como também o desenvolvimento de aplicações para processamento distribuído e de um Algoritmo Evolutivo. Para este último, pode-se verificar que, apesar da já esperada obtenção de soluções subótimas, a configuração de seus parâmetros é difícil, quando envolve um amplo espaço de busca como é o caso da configuração dos formigueiros em questão, com mais de 10^{70} combinações e sem conhecimento da função a ser avaliada, quantidade de máximos locais, espaçamento entre os mesmos, etc., sendo necessários diversos testes com variadas funções evolutivas conhecidas (em espaços de busca de mesmo tamanho) e, mesmo assim, sem qualquer garantia de que as soluções encontradas sejam próximas da ótima.

O software desenvolvido se mostrou eficiente e flexível para as simulações desejadas. A função antstep(), responsável pelo processamento da movimentação das formigas pode ser rapidamente adaptada para enviar comandos a robôs reais. Os dados do ambiente podem ser lidos se inseridos nas matrizes de obstáculos, comida, ninho e feromônios conforme são detectados por um sensor. Após as simulações necessárias, configurações criadas poderão ser totalmente portadas para os robôs, sem necessidade de um controle por um software central como o desenvolvido.

Este projeto é o único na literatura em que foi possível fazer com que as formigas encontrassem comida e retornassem ao ninho com a mesma sem o conhecimento da posição deste, além de fazerem, na maioria dos experimentos, trilhas próximas da ótima entre o ninho e a fonte mais próxima de comida. Este resultado é importante, pois poderá permitir a construção de robôs com custo menor, já que não será necessário que possuam

sensores de localização.

6.1. Trabalhos futuros

A ferramenta desenvolvida é de simples adaptação, já que todo o mecanismo de controle das formigas está contido em um único arquivo (antsalg.c), com funções que podem ser facilmente *interfaceadas* a partir de outros arquivos, como foi feito com a interface gráfica e o algoritmo evolutivo para execução em múltiplos processadores, que possuem o mesmo arquivo de controle, tendo alterações somente no arquivo principal. Este controle pode ser utilizado futuramente para modelagem e implementação de diferentes tipos de agentes, que poderão interagir ao mesmo tempo no ambiente. Por exemplo, podem ser criadas diversas colônias de formigas, que disputariam por recursos no ambiente.

Os parâmetros do ambiente simulado proporcionado pelo software podem ser configurados em robôs autônomos, permitindo assim que robôs reais possam obter o comportamento encontrado em simulação, com poucos sensores e baixo custo, o que pode ser útil em tarefas de exploração em ambientes desconhecidos ou perigosos para seres humanos.

Sugere-se também, como trabalho futuro, a execução do Algoritmo Evolutivo em um cluster com maior capacidade de processamento ou com mais tempo disponível, para obtenção de melhores resultados ao utilizar uma quantidade maior de testes e ambientes.

7. Referências Bibliográficas

[1] Gardner, M., The Game of Life, Part III, 1983.
[2] Dennett, Daniel, Darwin's Dangerous Idea: Evolution and the Meanings of Life, 1996.
[3] Liu, Yang e Passino, Kevin M. Swarm Intelligence: Literature Overview, The Ohio State University, 2000 - Forschungsbericht.
[4] Teuscher, Christof et al., Romero's Odyssey To Santa Fe: From Simulation To Real Life, The Ohio State University, 2000
[5] Y. Uny Cao et al., Cooperative Mobile Robotics: Antecedents and Directions, Kluwer Academic Publishers, Boston, 1997.
[6] Zhang, Y. et al., Evolving neural controllers for collective robotic inspection, Proceedings of the 9th Online World Conference on Soft Computing in Industrial Applications, 2004.

[7] Michel, O., Webot: Professional mobile robot simulation, Journal of Advanced Robotic Systems, 2004, 1(1):39–42.
[8] Nelson, A. L., Grant, E. e Henderson, T. C., Evolution of neural controllers for competitive game playing with teams of mobile robots, Robotics and Autonomous Systems, 2004b, 46:135-150.
[9] Miazaki, M. e Simões, E. D. V., Um Algoritmo Evolutivo para um Sistema de Controle de Navegação de Multi-Robôs Móveis Autônomos, SBRN 2004 - VIII Brazilian Symposium on Neural Networks, São Luís, Maranhão, 2004.
[10] Dorigo, M., V. Maniezzo, & Colorni, A., The Ant System: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics – Part B, 1996.
[11] Colorni, A., Dorigo, M., Maniezzo, V. e Trubian, M., Ant system for job shop scheduling. Belgian Journal Operations Research, 1994, v. 34.
[12] Shtovba, S., Ant algorithms: Theory and applications. Programming and Computer Software, 2005, v. 31 (no 4).
[13] Vittori, K. e Araújo, A. F. R. (2001). Agent-oriented routing in telecommunications networks. IEICE Transactions On Communications, E84-B(11):3006–3013.
[14] Roulston, T. H. e Silverman, J., The effect of food size and dispersion pattern on retrieval rate by the argentine ant, Linepithema humile (Hymenoptera: Formicidae), Journal of Insect Behaviour, 2002, 15(5):633–648.
[15] V. Calenbuhr & J.L. Deneubourg, A model for trail following in ants: Individual and collective behaviour. In Biological Motion, Eds. W. Alt and G. Hoffman. Lecture Notes in Biomathematics, Springer-Verlag, Berlin, 1990.
[16] Wilson, E. O., The insect societies, The Belknap Press of Harvard University Press, 1971.
[17] Deneubourg, J. L., Goss, S., Pasteels, J. M., Fresneau, D., e Lachaud, J. P., Behavior in Social Insects, 1987, v. 54.
[18] Dorigo, M., Optimization, learning and natural algorithms, Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.
[19] Tomassini, M., A Survey of Genetic Algorithms, Annual Reviews of Computational Physics, World Scientific, 1995.
[20] http://www.swarm.org/wiki/Swarm_main_page, acessado em Abril de 2007
[21] http://ccl.northwestern.edu/netlogo/, acessado em Abril de 2007.
[22] http://www.ai.mit.edu/projects/ants/, acessado em Abril de 2007.
[23] http://www.irobot.com/sp.cfm?pageid=149, acessado em Abril de 2007.
[24] Costa, D. N., Simulador extensível para navegação de agentes baseado em inteligência de enxames, Ph.D. thesis, Universidade de São Paulo, 2007.
[25] http://www.netlib.org/pvm3/, acessado em Maio de 2007
[26] http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page1905.htm, acessado em Março de 2007

Apêndice I

Tabela I. Melhores soluções obtidas pelo Algoritmo Evolutivo

Parâmetro\Ambiente	**Experimento 1**	**Experimento 2**	**Experimento 3**	**Experimento 4**	**Experimento 5**
Evaporate	0	4	0	8	1
each interactions	128	128	128	16	8
Saturation	2000000000	2000000000	16384	2000000000	4096
Turn prob 0º Turn prob 5º	27% 8%	35,6% 24%	49% 24%	91,8% 1%	42% 0,5%
Turn prob 10º	1%	0,2%	1%	4%	0,5%
Turn prob 15º	24%	16%	1%	1%	8%
Turn prob -5º	8%	0,2%	24%	2%	24%
Turn prob -10º	24%	24%	0%	0,2%	24%
Turn prob -15º	8%	0%	1%	0%	1%
1-Follow F1	68%	0,5%	99,8%	99%	99,8%
1-Threshold min	275	100	50	100	50
1-Threshold max	350	200	2000000000	200	25600
1-Against F1	1%	2%	0,2%	1%	2
1-Threshold min	12800	6400	1600	6400	150
1-Threshold max	800	3200	1600	0	20
1-Hard follow	True	False	True	True	True
1-Relative Thr	False	False	True	True	True
2-Follow F1	100%	98%	0%	25%	99%

2-Threshold min	2000000000	200	2000000000	12800	1600
2-Threshold max	6400	2000000000	3200	6400	0
2-Against F1	0%	2%	0,5%	68%	1%
2-Threshold min	25600	2000000000	800	275	350
2-Threshold max	10	3200	25600	350	3200
2-Hard follow	True	True	False	True	True
2-Relative Thr	True	True	True	True	True
3-Follow F1	96%	24%	100%	100%	68%
3-Threshold min	200	1600	200	150	200
3-Threshold max	9600	150	2000000000	2000000000	1600
3-Against F1	4%	0,2%	0%	0%	24%
3-Threshold min	25600	2000000000	1600	200	0
3-Threshold max	3200	100	10	150	200
3-Hard follow	True	True	True	False	True
3-Relative Thr	True	False	False	True	True
1-Deposit F1	4	4	4	16	16
1-Time limit	100	400	10	3200	100
1-Sensor radius	2	16	16	16	16
1-Sensor position	32	32	28	32	14
1-Behavior config	3	2	3	3	3
1-Prob change to state 4	1 em 2000000000	1 em 1600	1 em 1600	1 em 350	1 em 401
1-Prob change to state 5	1 em 1	1 em 2000000000	1 em 276	1 em 2000000000	1 em 1

2-Deposit F1	16	16	16	16	16
2-Time limit	275	3200	350	3200	25
2-Sensor radius	16	16	16	16	16
2-Sensor position	32	32	32	32	30
2-Behavior config	3	2	3	3	1
2-Prob change to state 3	1 em 6400	1 em 2000000000	1 em 351	1 em 12801	1 em 9601
3-Deposit F1	2	16	4	16	1
3-Time limit	1600	200	3200	3200	3200
3-Sensor radius	16	16	16	16	16
3-Sensor position	30	32	28	28	32
3-Behavior config	3	3	3	3	1
3-Prob change to state 2	1 em 12800	1 em 9600	1 em 2000000000	1 em 800	1 em 25600
4-Deposit F1	2	16	16	16	16
4-Time limit	275	275	3200	10	0
4-Sensor radius	16	2	16	16	0
4-Sensor position	30	32	28	30	32
4-Behavior config	3	3	1	3	3
4-Prob change to state 1	1 em 50	1 em 1	1 em 3200	1 em 6400	1 em 150
4-Prob change to state 5	1 em 350	1 em 101	1 em 1600	1 em 9600	1 em 6400
5-Deposit F1	2	16	16	4	16
5-Time limit	1600	25	275	3200	1600

5-Sensor radius	16	2	16	16	16
5-Sensor position	32	32	32	32	32
5-Behavior config	3	2	1	3	3
5-Prob change to state 1	1 em 25600	1 em 12800	1 em 100	1 em 3200	1 em 25600
5-Prob change to state 4	1 em 3200	1 em 100	1 em 10	1 em 350	1 em 276
Turn angle 0	150º	180º	160º	180º	0º
Turn angle 1	180º	180º	0º	30º	180º
Turn angle 2	20º	90º	20º	0º	180º
Turn angle 3	0º	20º	160º	10º	0º
Turn angle 4	180º	150º	180º	180º	180º
Turn angle 5	170º	180º	170º	180º	170º
Turn angle 6	10º	180º	180º	30º	10º
Turn angle 7	180º	180º	180º	180º	170º
Turn angle 8	180º	160º	180º	10º	20º
Turn angle 9	10º	180º	150º	180º	20º
Turn angle 10	30º	30º	30º	10º	150º
Turn angle 11	20º	170º	20º	180º	10º
Turn angle 12	90º	180º	150º	10º	10º
Turn angle 13	30º	180º	180º	180º	180º
Radiusdecay	1,000	0,985	1,000	1,000	0,650