Tarefa 1 - Explorar X Explotar e Escolher: Resgate de Vítimas de Catástrofes Naturais, Desastres ou Grandes Acidentes

Breno M. de Abreu [1561286]

Departamento de Informática (DAINF) - Universidade Tecnológica Federal do Paraná (UTFPR)

brenoabreu@alunos.utfpr.edu.br

Abstract. *This paper describes the development process of the project called Tarefa 1: Explorar X Explotar e Escolher, proposed by professor Cesar Augusto Tacla of the Intelligent Systems course. The project has the goal of applying the knowledge learned in the course about informed search, online search and genetic algorithms.*

Resumo. *Este artigo descreve o processo de desenvolvimento da Tarefa 1: Explorar X Explotar e Escolher, proposta pelo professor Cesar Augusto Tacla na disciplina Sistemas Inteligentes. O projeto tem por objetivo aplicar os conhecimentos aprendidos para a aplicação de algoritmos de busca informada, busca online e algoritmos genéticos.*

1. Introdução

O problema descrito na tarefa apresenta um ambiente 2D e separado em *tiles* quadrados onde há um conjunto de paredes e vítimas. Um agente chamado explorador irá, sem conhecimento prévio do ambiente, explorar o ambiente buscando mapear a localização de paredes e vítimas; e um outro agente, chamado socorrista, irá utilizar a versão do ambiente mapeada pelo explorador para tentar salvar as vítimas entregando-lhes suprimentos. Ambos os agentes têm um limite de ações que podem ser realizadas em seu turno, expressadas como medidas de tempo. O objetivo do explorador é mapear o máximo possível do ambiente dentro do tempo determinado, e o objetivo do socorrista é tentar salvar a maior quantidade de vítimas também dentro do limite de tempo determinado. As vítimas têm graus diferentes de criticidade que afetam a pontuação total de desempenho dos agentes. O desempenho é calculado no final da execução do programa.

Podemos dividir o problema em três subproblemas:

1. Cálculo da menor distância entre dois pontos no ambiente: é necessário descobrir a menor distância entre dois pontos para que seja possível gastar a menor quantidade de tempo possível atravessando o mapa. Isso se faz necessário devido à restrição de tempo determinada no problema. Ao resolver esse problema, evita-se dos agentes perderem tempo tomando caminhos mais longos e menos eficientes. É imprescindível para a solução do subproblema 3.

2. Exploração do ambiente sem informação prévia: como os agentes não conhecem o ambiente, se faz necessário aplicar um algoritmo que dê ao agente explorador a capacidade de descobrir quais as barreiras que existem no ambiente, além de mapear a localização das vítimas para que elas possam ser salvas pelo agente socorrista. Sem essa informação, o agente socorrista iria perder muito mais tempo tentando encontrar as vítimas, o que iria reduzir a eficiência da operação drasticamente.

3. Menor caminho possível entre um conjunto de pontos no ambiente: é necessário descobrir uma forma de visitar a maior quantidade os estados objetivos, nesse caso as vítimas, dentro do tempo determinado. Esse problema é conhecido como o problema do caixeiro viajante e sua solução nos permite visitar a maior quantidade de vítimas de uma maneira que não haja perda de tempo causada pela movimentação desnecessária no ambiente.

2. Fundamentação Teórica

Os tipos de busca vistas durante a disciplina foram:

- Busca cega: busca em largura, busca em profundidade e busca de custo uniforme
- Busca informada: busca gulosa e A*
- Busca online: DFS e LRTA
- Busca local: subida de encosta, têmpera simulada e busca em feixe local
- Algoritmos genéticos

Os tipos de busca pertinentes para esse projeto são: busca informada para a resolução do subproblema 1; busca online para a resolução do subproblema 2; e algoritmos genéticos para a resolução do subproblema 3.

3. Metodologia

O ambiente consiste em um *grid* 2D contendo *tiles* quadrados. O *grid* tem um valor de X *tiles* de comprimento por Y *tiles* de altura. Cada *tile* tem oito vizinhos, a não ser os *tiles* localizados nos limites do *grid*, que têm menos de oito vizinhos. Existem quatro tipos de *tiles*: o primeiro são posições vazias, onde os agentes podem se mover livremente; o segundo tipo são barreiras, que bloqueiam a passagem dos agentes; o terceiro são vítimas, que não bloqueia a passagem dos agentes e podem ser escaneados ou socorridos pelos agentes, são as posições objetivo do *grid*; e o quarto é a base, que é a posição inicial de um agente e onde deve retornar antes do tempo acabar.

Os estados do problema são exatamente os *tiles* dentro do *grid*. Cada posição representa um estado diferente e é possível ter um conjunto de X * Y estados, que podem ser algum dos quatro tipos de *tiles* descritos acima. Um *tile* pode ser de apenas um tipo entre os quatro. Os agentes mudam de estados ao mudar de um *tile* para outro, e cada ação que tomam gasta uma determinada quantidade de tempo para ser efetuada.

O ambiente é parcialmente observável, multi-agente, cooperativo, determinístico, sequencial, estático e discreto.

Os dois agentes percorrem o ambiente. O agente explorador tem a capacidade de mapear novos estados encontrados e escanear vítimas em busca do valor que representa sua criticidade. Isso deve ser feito até não haver mais tempo disponível para efetuar a exploração. O problema do agente explorador é caracterizado pelo subproblema 2. O agente socorrista irá utilizar as informações encontradas pelo explorador para encontrar o caminho que permite-lhe socorrer a maior quantidade de vítimas dentro do tempo determinado. O problema do agente socorrista é caracterizado pelo subproblema 3. Ambos os agentes precisam que o subproblema 1 seja resolvido.

Para resolver o subproblema 1 foi utilizado o algoritmo de busca informada A*. Dado que é possível calcular a heurística do problema utilizando a distância euclidiana entre dois pontos, o que permite nos informar o quão próximo o agente está de seu objetivo, a utilização do algoritmo A* se mostra a mais eficiente. Não foram utilizados algoritmos de busca cega pois não é possível encontrar a solução ótima se houver custo entre os estados, o que nesse caso há. Sendo que temos uma informação (a heurística) para cada estado, a solução de busca gulosa também poderia ser utilizada, mas como a movimentação dos agentes pode gastar tempos diferentes dependendo de para onde ele se movimenta, é necessário também considerar esse custo, e apenas o A* o considera enquanto que a busca gulosa apenas considera a heurística. A busca gulosa também não é ótima. Para o caso da busca de custo uniforme, quando comparado ao A* sabe-se que a última utiliza também uma heurística e a primeira apenas o custo. Dessa forma, com o A* é possível minimizar a quantidade de nós expandidos utilizando tanto o custo quanto a heurística. A busca de custo uniforme iria expandir mais nós que o A*.

Para resolver o subproblema 2 foi utilizado o algoritmo de busca online DFS. Dado que o ambiente é estático e o agente não conhece os efeitos de suas ações no ambiente. Além disso, as ações são reversíveis, o que permite utilizar o algoritmo. O agente precisa explorar o ambiente, ou seja, precisa conhecer o efeito de cada ação (movimentação) pelo mapa. Precisa saber que ao efetuar uma ação de movimentação em um *tile*, ele bate em uma barreira, ou encontra uma vítima, ou encontra um *tile* que permite que ele se mova livremente. Com isso é possível mapear o ambiente desconhecido. O algoritmo LRTA* não foi utilizado pois só pode ser utilizado em um ambiente conhecido e que também é semi-dinâmico. O ambiente em questão não apresenta nenhum desses casos.

Para o subproblema 3 foi utilizado algoritmos genéticos para solucionar o problema que pode ser caracterizado como o problema do caixeiro viajante. Esse é um problema de permutação que pode ser aproximado utilizando algoritmos genéticos. Os demais algoritmos de busca local não foram utilizados pois apenas conseguem resolver problemas de combinatória e não de permutação. Além disso, não utilizam nenhuma informação, como o *fitness* que é utilizado nos algoritmos genéticos para guiar a busca.

Para o agente explorador foi utilizado o algoritmo de busca online DFS. O agente explora livremente o ambiente apenas nas quatro principais direções: cima, baixo, direita e esquerda. As demais direções não foram utilizadas e nem testadas pelo autor. A sequência de ações que apresentou melhores resultados foi: baixo, direita, esquerda e cima. Quando encontra uma vítima, escaneia-a e guarda suas informações. A cada iteração a menor distância entre o agente e a base é calculada utilizando o algoritmo A*. Quando o tempo restante está próximo do limite do tempo que será gasto para retornar para a base, o agente retorna utilizando o caminho encontrado pelo A*.

Os dados sobre o ambiente são informados pelo agente socorrista que utiliza um algoritmo genético para encontrar a menor distância entre todas as vítimas de forma que saia da base e retorne para ela após socorrer a última vítima. O algoritmo gera 200 gerações, utilizando 10 indivíduos como amostra e utilizando 6 desses para gerar mais 6 indivíduos através de *crossover*. Os indivíduos são sorteados aleatoriamente no início e seus *locus* são as vítimas que serão visitadas na sequência em que se apresentam no vetor, ou seja, a vítima na posição 1 será visitada primeiro, e a vítima na posição N será a enésima vítima a ser visitada. Há 5 por cento de chance de haver mutações e 80 por cento de haver *crossover.* Caso não seja possível salvar todas as vítimas no período de tempo determinado, a vítima com menor criticidade que está mais distante da base será descartada, e não será socorrida, o cálculo então é realizado novamente com todas as vítimas menos a que foi descartada. O processo é repetido até que o processo possa ser realizado dentro do tempo. Para executar o algoritmo genético é necessário utilizar o A* para encontrar as menores distâncias entre cada *locus* presente na sequência de vítimas. O algoritmo calcula o *fitness* de um indivíduo somando a menor distância entre cada vítima como apresentada na sequência determinada aleatoriamente. É considerado que o primeiro e o último *locus* é a posição da base. O algoritmo utiliza a seleção por roleta para selecionar os indivíduos para *crossover*. Para a mutação é apenas trocada a posição de dois *locus* aleatórios.

4. Resultados e análise

Para o espaço de estados compartilhado pelo professor, o agente explorador conseguiu encontrar ~57% das vítimas gastando ~25 medidas de tempo por vítima. O percentual ponderado de vítimas encontradas foi de ~55% em média.

O agente socorrista conseguiu socorrer todas as vítimas encontradas pelo explorador, tendo o percentual de vítimas socorridas (em relação ao total de vítimas no ambiente) de ~57%, gastando ~6 medidas de tempo por vítima. O percentual ponderado de vítimas socorridas foi de ~55%, tendo uma quantidade considerável de tempo sobrando ao retornar para a base.

5. Conclusão

A solução não conseguiu encontrar e socorrer todas as vítimas. O autor acredita que o agente socorrista conseguiria socorrer todas as vítimas, ou pelo menos a maior parte delas e as mais críticas, se o agente explorador tivesse feito um trabalho melhor de encontrar as vítimas. O algoritmo de exploração poderia ter sido mais bem elaborado para considerar as demais direções que não foram implementadas.

Além disso, algumas implementações não foram idealmente elaboradas previamente, gastando mais tempo do que o necessário para serem executadas. Um gargalo encontrado foi na implementação de um algoritmo que calcula a distância entre uma vítima e todas as demais. Considerando um espaço de estados grande, com muitas vítimas, percebe-se que o cálculo dessa tabela auxiliar poderia levar um tempo considerável. Talvez fosse melhor calcular a distância apenas na hora que fosse realmente necessária. O autor não realizou essa modificação para verificar se a performance seria melhorada.

Modificações para resolver os problemas acima podem ser implementadas no futuro.

A solução é enviesada pois descarta a possibilidade de socorrer vítimas consideradas como menos críticas caso perceba que não há tempo para salvar todas as vítimas. Caso a modelagem para definir a criticidade de uma vítima seja feita de uma maneira antiética, é possível que a solução irá desconsiderar vítimas que deveriam ser salvas mas que foram descartadas pelo algoritmo. A modelagem do programa afeta as vítimas da seguinte forma: dependendo da estratégia utilizada para explorar o ambiente, é possível que certas vítimas nunca sejam encontradas e não sejam salvas. Da mesma forma, caso a estratégia de encontrar o caminho entre as vítimas seja implementada de forma não eficiente, vítimas podem não ser salvas dentro do tempo, e vítimas com menor criticidade serão descartadas e não serão socorridas.

6. Referências Bibliográficas

Tacla, C. “Agentes e Ambientes”. Slides.

Tacla, C. “Busca Cego ou Sem Informação”. Slides.

Tacla, C. “Busca Informada”. Slides.

Tacla, C. “Busca Online”. Slides.

Tacla, C. “Busca Local (Sem Informação)”. Slides.

Tacla, C. “Algoritmos Genéticos”. Slides.

Kie Codes. “Genetic Algorithm Explained by Example”. YouTube. https://www.youtube.com/watch?v=uQj5UNhCPuo

Yarpiz. “Introduction to Genetic Algorithms - Practical Genetic Algorithms Series”. YouTube. https://www.youtube.com/watch?v=Fdk7ZKJHFcI

7. Apêndice

Como executar o código:

1. Dentro da pasta Resources é necessário copiar os dois arquivos `ambiente.txt` e `sinais_vitais.txt`. É necessário que tenham exatamente esses nomes e estejam formatados da maneira como estão nos arquivos básicos.
2. Abrir o terminal na pasta onde está o arquivo `main.py`.
3. No terminal, escrever `python main.py`
4. Pressionar Enter.

Nenhum argumento adicional é necessário.

O programa irá imprimir o mapa original e o mapa explorado pelo agente explorador. Irá imprimir também os resultados de desempenho como requeridos no arquivo que contém o enunciado para o problema.