Agentes para localização e salvamento
Escolher e Deliberar

Gustavo Henrique Zeni[1], **Ianca Polizelo**[1]

[1]Engenharia de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)

gustavozeni@alunos.utfpr.edu.br, iancapolizelo@alunos.utfpr.edu.br

1. Introdução

O trabalho tem como objeto principal resgatar vítimas de catátrofes naturais, desastres ou grandes acidentes. Para tal, faz-se uso de dois agentes, um para vasculhar o ambiente e construir um mapa que conterá a localização das vítimas, outro para socorrer as vítimas localizadas.

Foi implementado anteriormente um agente vasculhador que tinha como objetivo percorrer um ambiente desconhecido a fim de construir um mapa do mesmo, que poderia conter: espaços vazios, espaços com obstáculos e espaços com vítimas. Junto com construção do mapa, o agente teve que localicar as vítimas e ler seus sinais vitais. Os sinais vitais seriam utilizados por outro agente para que fosse possível realizar o cálculo da dificuldade de resgate da vítima. O vasculhador ainda possuía um tempo limite em que poderia explorar o ambiente.

Ainda na implementação anterior, foi realizado um segundo agente, o socorrista, que recebia as informações de mapa do vasculhador e calculava quais vítimas iria salvar. Para tal resgate, foi então escolhido salvar o maior número de vítimas dentro do tempo disponível, sem levar em conta sinais vitais.

Para este trabalho, será necessário implementar um algoritmo de busca que irá selecionar quais vítimas irá socorrer, de tal forma que a soma da gravidade dessas vítimas seja maximizada.

Para tal, o agente socorrista precisa receber do agente vasculhador a localização de cada vítima, assim como seus sinais vitais e também os dados de dificuldade de resgaste. Este agente terá um tempo limite *Ts* para conseguir salvar as vítimas e cada vítima possuirá um tempo *ti* de resgate individual. Logo, a seleção das vítimas salvas deve ser tal que a soma de todos os *ti*s seja menor ou igual a *Ts*.

O agente vasculhador utilizado para este trabalho será o mesmo construído para o anterior.

Os dois agentes recebem as seguintes informações: número máximo de linhas e colunas do ambiente a ser explorado, número total de vítimas, tempo máximo para o agente vasculhador localizar as vítimas, tempo máximo para agente socorrista salvar as vítimas e capacidade de carregamente de pacotes do agente socorrista.

Para esta tarde do trabalho, uma vítima é considerada salva quando ela é selecionado pelo agente socorrista.

2. Fundamentação Teórica

Para a realização deste trabalho foram vistos os seguintos tipos de busca:

- **Busca Cega:** são as buscas que não possuem nenhuma informação adicional sobre os estados além dos que já foram fornecidos na definição do problema. Aqui, existem algumas estratégias diferentes de busca, porém elas diferem entre si apenas pela ordem em que os nós são expandidos.
  - **Busca em Largura(Breadth-First):** é a estratégia em que o nó raiz é expandido primeiro, depois todos os nós sucessores do nó raiz, depois os sucessores desses outros nós e assim por diante. Logo, todos os nós de um nível serão expandidos antes de se expandir os nós dos níveis seguintes.
  - **Busca de custo uniforme:** nesta busca, ao invés de expandir o nó mais raso, será expandido o nó *n* com o *custo de caminho g(n) mais baixo*.
  - **Busca em profundidade(Depth-first):** é a estratégia que irá expandir o nó mais profundo na borda atual da ávore de busca. Assim, a busca prossegue até o nível mais profundo da ávore de busca, onde não existem mais sucessores para os nós, então o nó é retirado da borda e a busca volta para o nó seguinte.
  - **Busca em profundidade limitada:** aqui é definido um limite de profundidade pré-determinado, para contornar o problema de árvores com profundidade infinita.
  - **Busca de aprofundamento iterativo:** é uma mistura das duas buscas em profundidade anteriores para encontrar o melhor limite de profundidade. Isto é feito aumentando gradualmente o limite de profundidade até encontrar um objetivo.
  - **Busca bidirecional:** a estratégia aqui é realizar duas buscas simultâneas, uma partindo do estado inicial e outra partindo do objetivo, fazendo com que se encontrem no meio do caminho.
- **Busca Informada/heurística:** diferente da busca cega, a busca informada utiliza conhecimento de um problema além da definição. Neste tipo de busca, a seleção de qual nó será expandido será através de uma *função de avaliação f(n)*. Esta função é analisada como uma estimativa de custo, assim, o nó com a menor avaliação será o expandido.
  - **Busca Gulosa de Melhor Escolha:** é a estratégia que busca expandir o nó que está mais próximo de acordo com a heurística, com a intenção de que isso possa conduzir a uma solução rapidamente. O nó é avaliado usando apenas a função heurística, ou seja, $f(n) = h(n)$.
  - **Busca A*:** esta estratégia avalia os nós através da combinação de $g(n)$, o custo para alcançar o nó, e $h(n)$, o custo para ir do nó ao objetivo: $f(n) = g(n) + h(n)$. Assim, $f(n)$ será o custo estimado da solução de menor custo através de n. Esta é a busca mais conhecida dentre as buscas informadas.

- **Busca Local:** a busca local é utilizada quando não é relevante o caminho para se chegar ao objetivo, mas apenas de fato a solução encontrada. Assim, é possível utilizar algoritmos de busca local para problemas de otimização, onde se busca uma solução ótima para o problema. Também pode ser utilizado em espaços de estados muito grandes, pois não precisam de memória para guardar o caminho.

- **Busca de subida de encosta:** ele é um laço repetitivo que se move de forma contínua no sentido do valor crescente, ou seja, encosta acima. O algoritmo termina quando alcança o ”pico”em que nenhum vizinho tem valor mais alto. O algoritmo não mantém uma árvore de busca e, assim, a estrutura de dados do nó atual só precisa registrar o estado e o valor de sua função objetivo.
- **Têmpera simulada:** um algoritmo de subida de encosta que nunca faz movimentos em direção a estados com valores mais baixos pode acabar ficando preso em um máximo local, e não achar o máximo global. Já um algoritmo de subida com percurso aleatório encontraria o máximo global, porém seria muito ineficiente. Assim, o algoritmo de têmpera simulada busca combinar essas duas abordagens. Para essa busca, existe um laço de repetição mais interno muito parecido com o laço da subida de encosta, porém escolhendo movimentos aleatórios ao invés do melhor movimento. Caso o movimento melhore a situação, ele sempre será aceito, caso contrário, o algoritmo aceitará o movimento com alguma probabilidade menor que 1.
- **Busca em feixe local:** este algoritmo mantém o controle de k estados ao invés de apenas um, como mas buscas anteriores. Ele começa com k estados gerados aleatoriamente e vão sendo gerados todos os sucessores de todos os k estados conforme o algoritmo avança. Caso algum desses estados for um objetivo, o algoritmo para, senão ele selecionará os melhores sucessores a partir da lista completa e repetirá o procedimento. Dentro desta busca, temos uma variante chamada **busca em feixe estocástica**, onde, ao invés de escolher o melhor k dentro do conjunto de sucessores, esta busca escolhe k sucessores de forma aleatória, fazendo com que a falta de diversidade entre os k estados seja contornada. Pode ser feito uma analogia em relação a seleção natural através da reprodução assexuada, onde um único indivíduo é modificado para gerar um novo.
- **Algoritmos genéticos:** esta busca é uma variante da busca em feixe estocástica na qual os estados sucessores são gerados pela combinação de dois estados pais, ao invés de serem gerados pela modificação de um única estado. A analogia em relação à seleção natural é a mesma que se dá para a busca anterior, porém agora estamos lidando com uma reprodução sexuada, com troca de informação entre indivíduos, e não uma reproção assexuada, sem a troca.

3. Metodologia

O ambiente do problema pode ser caracterizado como:

- **determinístico**, pois o próximo estado é determinado pelo estado atual e pela ação executada pelo agente.
- **discreto**, pois o ambiente possui um número finito de estados.
- **semi-dinâmico**, pois apesar do ambiente não mudar geograficamente, ainda possue o fator do tempo contra os agentes.
- **colaborativo**, pois mesmo os agentes tendo objetivos diferentes, se o vasculhador não conseguir cumprir o seu objetivo, o socorrista também não irá conseguir.

- **Parcialmente observável**, pois o vasculhador não conhece o ambiente e vai descobrindo ao longo do processo, e depois passa para o socorrista, porém se o vasculhador não conseguir explorar todo o ambiente irá passar informação faltante para o socorrista, assim não podemos garantir que para o socorrista será totalmente observável.

Descrevemos formalmente o problema a seguir:

- **Estados:** O estado é determinado pelas posições do agente, das paredes e das vítimas. Sendo que o agente pode estar em qualquer posição que não contenha paredes. Sua posição pode conter ou não uma vítima. Assim, o tamanho do espaço de estados sera: $(maxLin * maxCol)((2^{maxLin*maxCol} - p)$, onde p é o número de paredes existentes no ambiente, o primeiro parênteses corresponde ao número de espaços que tem no ambiente e o segundo parênteses corresponde ao número de estados que pode existir em cada espaço, sendo que podem ser 2, vazio ou com vítima. Os espaços com paredes tem que ser desconsiderados, pois não podem mudar de estado, serão sempre ”parede”.
- **Estado inicial:** os agentes começam nas coordenadas [0,0]
- **Ações:** os agentes podem se mover para cima, baixo, direita, esquerda e diagonais. Também podem ler sinais vitais das vítimas, recarregar a bateria e carregar suprimentos.
- **Teste de objetivo:** para o agente vasculhador será mapear todo o ambiente localizando todas as vítimas e paredes, para o agente socorrista será resgatar o maior número de vitímias possíveis de forma a maximizar a soma das gravidades levando em conta que a soma dos tempos de resgate individuais não ultrapasse o tempo total de resgate que o agente possui.
- **Modelo de transição:** as ações tem seus efeitos esperados, a não ser as ações: mover para esquerda quando houver uma parede na esquerda, mover para direita quando houver uma parede na direita, mover para baixo quando houver uma parede abaixo, mover para cima quando houver uma parede acima e mover para diagonal quando houverem duas paredes entre o estado atual e a diagonal.
- **Custo de caminho:** os passos para esquerda, direita, cima e baixo tem custo 1, e os passos para a diagonal tem custo 1,5.

Dentro deste problema apresentado, o vasculhador projetado realiza uma busca on-line, **busca em profundidade**, para mapear o ambiente e localizar as vítimas, lendo os sinais vitais das vítimas. Depois que obtem todos as informações possíveis, calcula o melhor caminho para voltar para a base, através de uma **busca gulosa**, e passa o que coletou para o socorrista. Tudo isso levando em consideração o tempo de exploração que o agente possui. Para esta parte do problema foi utilizada a mesma implementação do trabalho anterior.

O socorrista recebe as informações coletadas pelo vasculhador e realiza uma busca local através de **algoritmo genético através de crossover** para que possa maximizar a soma da gravidade das vítimas resgatadas. O agente também leva em conta a soma dos tempos de resgates individuais, para que a soma deles não ultrapasse o tempo total de resgate que ele possui. Esta forma foi a escolhida para que pudesse ser utilizada a vantagens de manter a diversidade da população e ainda levando em conta a troca de informações entre as gerações criadas, com a intenção de se encontrar a melhor geração possível.

4. Resultados e Análise:

O cálculo de desempenho é feito utilizando as sequintes equações:

- **Valor acumulado dos gi das vítimas escolhidas para salvamento:**
  $G = \sum_{j=1}^{V_s} g_i = 4,83$, tal que $\sum_{j=1}^{V_s} t_i \leq T_s$

- **Número de vítimas salvas pelo tempo gasto:**
  $S = \frac{V_s}{t_s} = \frac{12}{395} = 0,030$

- **Número de vítimas salvas em 5 extratos de gravidade pelo tempo gasto**

  -]0.8, 1]: 3
  -]0.6, 0.8]: 1
  -]0.4, 0.6]: 1
  -]0.2, 0.4]: 1
  - $]0, 0.2]$: 6

  $Se = \frac{5V_sg5+4V_sg4+3V_sg3+2V_sg2+V_sg1}{ts(5V_sg5+4V_sg4+3V_sg3+2V_sg2+V_sg1)} = \frac{5.3+4.1+3.1+2.1+1.6}{395(5.3+4.1+3.1+2.1+1.6)} = \frac{30}{395.30} = 2,5$

Para chegarmos a esses valores foram utilizados os seguintes parâmetros: população inicial igual a 10, porcentagem de chance de crossover igual a 70%, porcentagem de mutação igual a 20% e número de gerações igual a 100. Também foi utilizado como função fitness o somatório das gravidades individuais e a codificação sendo $vítima- > \{0, 1\}$.

A abordagem escolhida pela equipe para soluções que excedessem o tempo estimado foi punitiva, ou seja, a todas que excederam foi atribuido um fitness de 0, garantindo que não seriam escolhidas para as próximas gerações.

Para a escolha dos pares do crossover, foi feito uma normalização do fitness de todas as soluções daquela geração de forma que a soma total de todos os fitness fosse igual a 1. Em seguida, foi gerado um número aleatório entre 0 e 1 e, ao iterar a geração, o fitness "acumulado"foi sendo somado ao longo dos cromossomos (vetor de vítimas selecionadas para fazer parte da solução), e o primeiro cromossomo cujo fitness acrescentado do fitness acumulado fosse maior do que o valor aleatório era selecionado para compor o vetor de pais.

A equipe também optou por adicionar a seleção elitista ao problema, ou seja, a cada mudança de geração, as duas melhores soluções da geração atual eram simplesmente passadas para a próxima, a fim de garantir que as próximas gerações sejam necessariamente melhores do que as anteriores.

Os parâmentros citados acima sobre aumentado para fins de testes e comparações com os valores, porém notou-se que os resultados práticos obtidos, número de vítimas salvas e gravidade acumulada, não se alterava significativamente, mas, com os parâmetros aumentados, o código demorou muito mias para rodar do que com os parâmetros originais.

5. Conclusões

Para a solução deste trabalho foi utilizado parte da implementação do código feito para a solução do trabalho anterior, o que ajudou muito na economia de tempo e também para que fosse possível focar em estudar e implementar um novo tipo de busca, neste caso a busca local. Também, visto que parte do código foi reaproveitada, algumas das dificuldade encontradas no trabalho anterior não foram um impedimento no desenvolvimento deste trabalho, pois o entendimento do código disponibilizado anteriormente pelo professor já tinha sido cumprida.

O desenvolvimento deste trabalho agregou ainda mais para o conhecimento já adquirido anteriormente, pois é algo complementar ao que já foi feito anteriormente, aumentando ainda mais o repertório dos autores sobre buscas como forma de resolução de problemas.

Comparativamente com a implementação do trabalho 1, esta se desenvolveu melhor, pois foi possível chegar ao objetivo final dentro do prazo definido para entrega do trabalho.

Na primeira data de entrega, o código apresentava baixíssima variabilidade e estagnação desde a primeira geração, apesar das mutações. A equipe então descobriu que o problema advinha de um bug em python, no qual ao fazer uma cópia de um dict (dicionário), mesmo usando a função própria para copiar (e não criar uma referência) .copy(), a cópia gerada era, mesmo assim, uma referência. Isso foi consertado usando a biblioteca deepcopy, que ao invés de criar uma cópia da forma comum em python, cria uma cópia de literalmente tudo e todos os campos do dict, desde os principais até os mais internos.

Ao final de todas as gerações, a última geração é iterada e a solução com o melhor fitness é selecionada como a melhor solução. Após debate, a equipe chegou à conclusão de que, caso haja empate, a melhor abordagem de desempate seria favorecer a solução que possua um maior número de vítimas em estado mais grave, mas infelizmente não foi capaz de implementar essa adição no tempo hábil de entrega do projeto.

Referências

Russell, S. and Norving, P. (2013). *Inteligência Artificial*. Elsevier Editora Ltda, 3th edition.

Tacla, C. A. *Tarefa 2: Escolher/Deliberar*. `https://moodle.dainf.ct.utfpr.edu.br/mod/assign/view.php?id=46533`.

Tacla, C. A. and Klein, L. C. *RescueSimulator*. `https://github.com/peteco-utfpr/RescueSimulator`.

[Russell and Norving 2013]

[Tacla]

[Tacla and Klein]

6. Apêndice:

Os códigos para este projeto podem ser encontrados no link abaixo:

https://github.com/Cheddargt/sist-intel