Inteligência Artificial 2021/22

Numbrix – Grupo 19

1. Solução Proposta

O objetivo deste projeto é desenvolver um programa que resolva o problema Numbrix, de forma eficiente, utilizando as técnicas de procura lecionadas nas aulas teóricas, por isso, foi crucial reduzir ao máximo o fator de ramificação da árvore de procura. Para tal, foram implementadas diversas ideias para reduzir, sempre que possível, o número de ações redundantes ou incorretas num dado nó da árvore:

1. **Fixar um número por nível da árvore de procura:** apenas são geradas as ações para um número em cada chamada à função "*actions*". Devido à propriedade de comutatividade das ações do problema Numbrix, temos a garantia de não perder a completude das procuras.
2. **Escolher o número com menos ações possíveis:** à semelhança da heurística MRV, no contexto dos problemas de satisfação de restrições, é benéfico escolher o número com menos ações possíveis para uma dada instância de tabuleiro pois, desta forma, os algoritmos de procura conseguem eliminar caminhos impossíveis o mais cedo possível.
3. **Usar a distância de Manhattan:** sempre que a distância de Manhattan entre a célula onde se está a considerar colocar um dado número e a célula do número numericamente mais próximo já colocado no tabuleiro excede a diferença numérica entre estes dois números, temos a garantia de que esta ação leva a uma configuração de tabuleiro impossível de resolver. Temos esta garantia, uma vez que a distância de Manhattan corresponde ao número de células no caminho mais curto entre estas duas posições, que, consequentemente, corresponde ao número de números diferentes que teriam de ser utilizados para conectar estas duas posições.
4. **Validar posições:** é feita uma verificação com base no número de zeros e vizinhos numericamente adjacentes a uma célula, de forma a averiguar se uma dada posição permanece válida de acordo com as restrições do jogo e se não leva a uma configuração de tabuleiro impossível de resolver. Esta verificação é efetuada não só para a célula da ação a ser considerada, mas também para todas suas células adjacentes.
5. **Detetar quando existe um número impossível de colocar no tabuleiro:** sempre que existe um número cujo antecessor ou predecessor (n-1 e n+1, respetivamente) já se encontra no tabuleiro e que não tenha posições válidas, temos a garantia de que esta configuração de tabuleiro é impossível de resolver, pois, se o número não pode ser colocado a esta profundidade de procura, então vai ser impossível de o colocar no tabuleiro a qualquer outra profundidade superior.
6. **Detetar quando existe um quadrado sem número possível:** sempre que existe um quadrado completamente rodeado de células já preenchidas e que não pode conter nenhum dos números restantes dadas as restrições do jogo, temos a garantia de que esta configuração de tabuleiro é impossível de resolver.

2. Heurística Proposta

A ideia base da heurística usada nas procuras Greedy e A* é favorecer tabuleiros onde as células não preenchidas estão o mais próximas possível entre si. Desta forma, estes algoritmos de procura conseguem eliminar caminhos impossíveis a profundidades mais baixas, uma vez que, seguindo esta heurística, as restrições para os números ainda não colocados no tabuleiro são maximizadas. A computação do valor desta heurística, dada uma instância de tabuleiro, consiste em subtrair ao número de zeros restantes no tabuleiro (zeros(board)) a soma do número de zeros adjacentes a todos os zeros do tabuleiro (adjzeros(board)). Chegando assim, à fórmula final:

$$h(board) = max(0, zeros(board) - \alpha \cdot adjzeros(board)) \; \alpha \in]0, 1]$$

Esta fórmula conduz a uma heurística admissível uma vez que $0 \leq h(board) < nzeros$ e, no problema Numbrix, nzeros corresponde sempre à distância restante para se chegar à solução, caso esta exista.

3. Avaliação da Solução

Para avaliar a solução proposta e os diferentes tipos de procura foram medidos tempos de execução, número de nós expandidos e o número de nós gerados, para cada uma das principais procuras: Breadth-First Search (BFS), Depth-First Search (DFS), Greedy, e A*.

Para além dos inputs públicos fornecidos, foram, também, adicionados outros dois inputs de dificuldade acrescida: "*big1.txt*" e "*big2.txt*", de tamanhos 10 × 10 e 12 × 12, e com número inicial de células preenchidas 25 e 27, respetivamente. Desta forma é, não só, possível obter uma melhor comparação entre os algoritmos de procura mais eficientes, mas também avaliar diferentes variações da heurística proposta.

Todos os testes de medição de tempo de execução foram efetuados num computador com um processador Intel i7-6700 e 8 GB de memória RAM.

4. Análise dos Resultados Obtidos

O problema Numbrix apresenta algumas particularidades que podemos usar para tirar conclusões sobre as propriedades das diferentes procuras. Em primeiro lugar, é de notar que este problema tem um espaço de procura finito, uma vez que a profundidade máxima das árvores de procura corresponde ao número de zeros presentes no tabuleiro inicial. Em segundo lugar, como o custo de todas as ações é unitário, se houver mais do que uma solução para o problema, temos a garantia de que todas as soluções têm o mesmo custo. Por isso, podemos concluir que, no contexto deste problema, todas as procuras consideradas: BFS, DFS, Greedy e A* são completas e ótimas, pelas razões mencionadas acima, respetivamente.

Analisando os gráficos 2 e 3, podemos observar que tanto a BFS e a A* apresentam constantemente valores superiores em relação às procuras DFS e Greedy. Esta diferença já é bastante mitigada pelo facto das ações estarem a ser fortemente reduzidas na sua geração e, consequentemente, tal como se pode verificar pelo gráfico 1, as diferenças em tempos de execução são notáveis, mas nunca drásticas.

Comparando as duas melhores procuras, no gráfico 2, notamos que, em geral, a DFS apresenta valores de nós expandidos ligeiramente superiores em relação à Greedy, com exceção no "*input2.txt*" que, apesar de consistir num tabuleiro de tamanho relativamente pequeno (6 × 6), a procura DFS apresenta valores sete vezes maiores. Neste caso, a exclusão de ações e a ordem pela qual estas são geradas não chegam para guiar a procura DFS, fazendo com que esta procura escolha caminhos errados muito cedo e só comece a retroceder na árvore de procura em profundidades muito elevadas.

De seguida, observando os gráficos 4, 5 e 6, podemos concluir que, de facto, existe uma grande diferença de desempenho entre as duas procuras, principalmente no input "*big2.txt*", e que a heurística proposta tem um efeito muito positivo no desempenho da procura.

Finalmente, para aferir qual o melhor valor de α a usar na heurística proposta, foram feitas as mesmas medições para o input "*big1.txt*" para valores de α entre 0 e 1 em incrementos 0.1. Conclui-se então, através dos gráficos 7, 8 e 9, que o melhor valor para este parâmetro é 0.3, pois é este que minimiza todas as métricas de desempenho. Este foi o valor utilizado na versão final do projeto e, para além disso, foi também usado em todos os outros gráficos mencionados acima.

Concluindo, contrariamente ao que esperávamos, a implementação eficiente do nosso programa dependeu muito mais da remoção de ações inválidas e redundantes do que da criação de uma boa heurística. Pensamos que a razão para tal deve-se ao facto de, neste problema em específico, não existir um estado objetivo conhecido previamente, fazendo com que não seja possível criar heurísticas que estimem com muita precisão a distância ao nó objetivo. Por isso, pensamos que este problema é um bom candidato para ser abordado como um problema de satisfação de restrições, pois não só a natureza do problema é construída em restrições explícitas, mas, também, o seu estado é de representação simples e uniforme.

5. Anexos

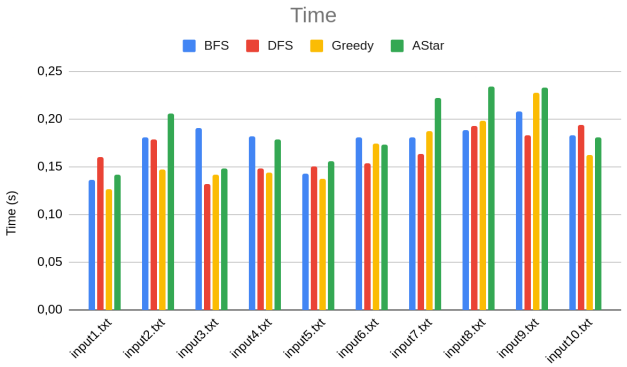

Gráfico 1 - Tempos de execução de todas as procuras em todos os inputs fornecidos

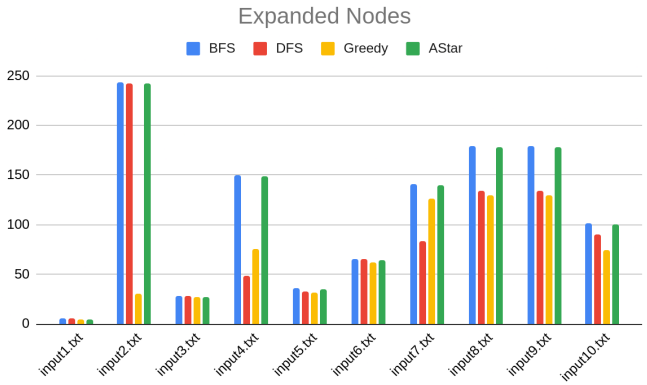

Gráfico 2 - Número de nós expandidos de todos as procuras em todos os inputs fornecidos

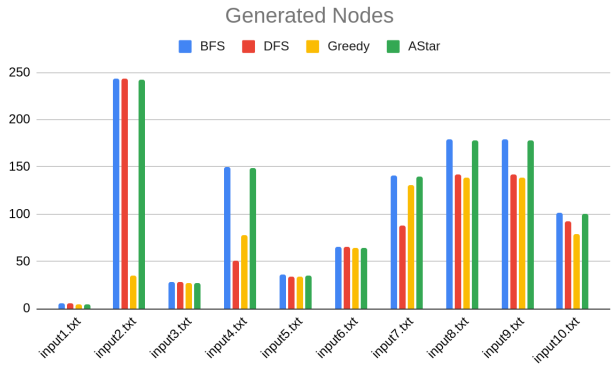

Gráfico 3 - Número de nós gerados de todas as procuras em todos os inputs fornecidos

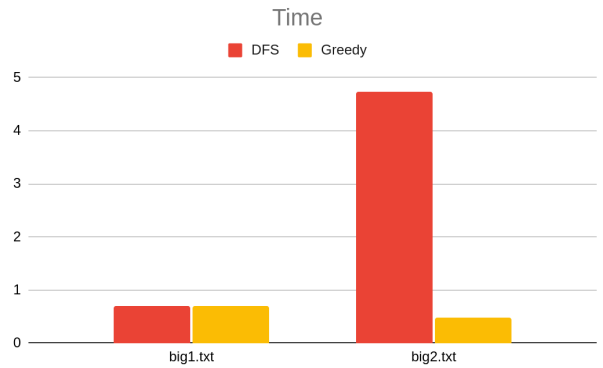

Gráfico 4 - Tempo de execução das procuras DFS e Greedy nos inputs de dificuldade acrescida

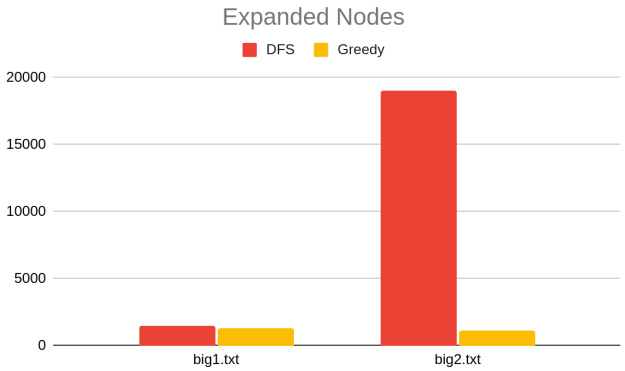

Gráfico 5 - Número de nós gerados das procuras DFS e Greedy nos inputs de dificuldade acrescida

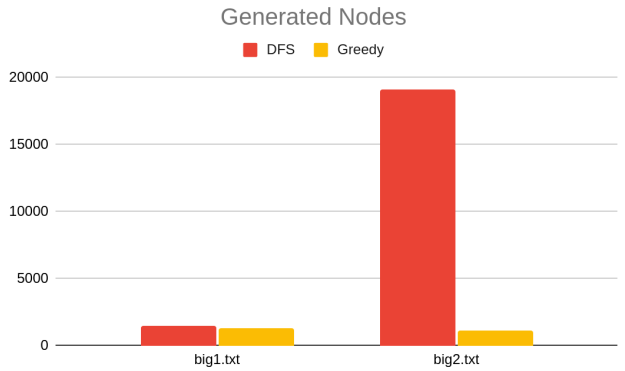

Gráfico 6 - Número de nós gerados das procuras DFS e Greedy nos inputs de dificuldade acrescida

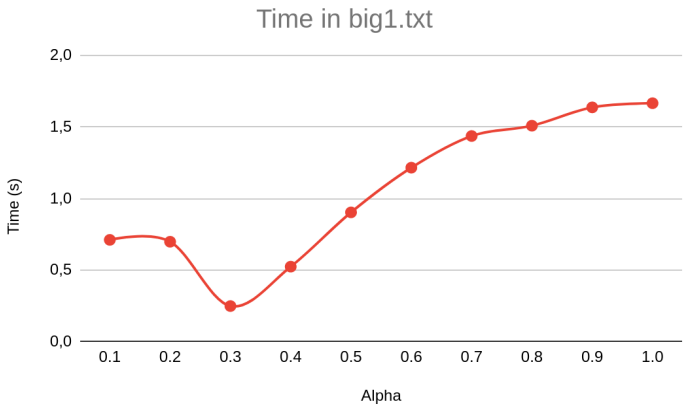

Gráfico 7 - Tempo de execução para diferentes valores de alpha da heurística proposta, usando a procura Greedy no input "*big1.txt*"

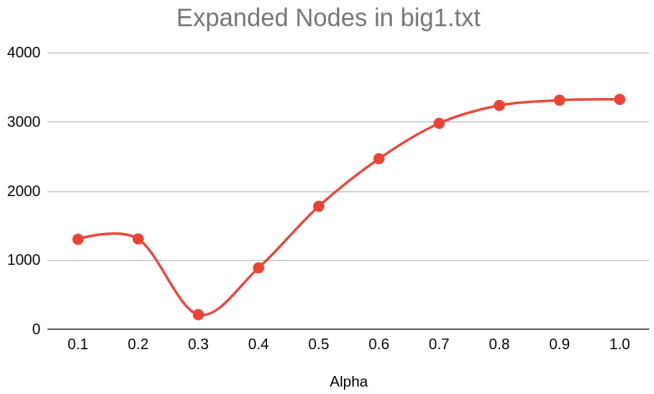

Gráfico 8 - Número de nós expandidos para diferentes valores de alpha da heurística proposta, usando a procura Greedy no input "*big1.txt*"

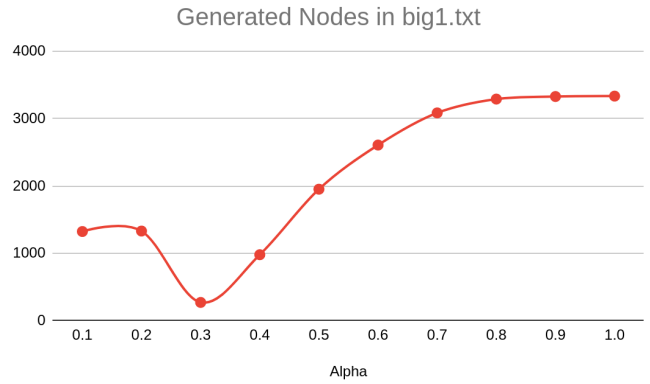

Gráfico 9 - Número de nós gerados para diferentes valores de alpha da heurística proposta, usando a procura Greedy no input "*big1.txt*"