Inteligência Artificial 2021/22 (P4) Grupo al027
Relatório de Projeto: Takuzu 91110 Inara Parbato
97375 Alexandra Pato

Descrição da Solução:

Lê-se o input do stdin, traduzindo-o para uma instância de *Board (board)*, pela função parse_instance_from_stdin. O *board* é uma representação de um problema takuzu.

Um *Board* tem os atributos:

- *size*, inteiro que representa o número de linhas e colunas;
- *rows*, uma lista de listas, e a sua transposta, *cols*, tal que *rows*[i][j] e *cols*[j][i] representam ambos o número da linha i, coluna j (*cols* foi criada para facilitar verificações na vertical); *maxNrsPerLine*, inteiro que corresponde ao máximo número de 0's ou 1's que podem estar em cada linha (*ceiling* de *size*/2);
- *nrMissing*, inteiro que corresponde ao número de posições por preencher;
- *rowStatus*, dicionário que contém o estado atual das linhas de *rows* guardado em 3 entradas na forma de listas: "*Zeroes"* (lista do número de 0's por linha), *"Ones"* (lista do número de 1's por linha), e *"Missing"* (lista do número de posições por preencher por linha);
- *colStatus*, dicionário semelhante a *rowStatus* mas para as colunas de *cols.*

Após a geração de um *TakuzuProblem takuzu* a partir deste *board*, efetua-se uma procura em profundidade (*depth_first_tree_search(takuzu)*) com o objetivo de encontrar a solução do takuzu. Para tal, para cada estado, calcula-se quais as ações possíveis (*actions()*). A partir de um certo estado, existem ações "possíveis" (de acordo com o *board* do estado atual, é possível colocar o valor v na posição [i][j] sem quebrar imediatamente as restrições do problema), ações "impossíveis", e ações "certas" (de acordo com o *board* do estado atual, para uma dada posição, apenas é possível colocar o valor v na posição [i][j] sem quebrar imediatamente as restrições do problema). Em *actions()*, procura-se devolver uma única ação "certa" assim que esta é encontrada – e, à falta de uma, apenas duas ações "possíveis" mas não "certas" para uma mesma posição. Tudo é efetuado, sempre, pela mesma ordem:

1. Ações "certas" (relembrar que, se se encontrar uma ação numa sub-secção, devolve-se essa e apenas essa imediatamente, não se passando nem pelas sub-secções seguintes nem pela secção 2.):
    a. Para cada linha não preenchida: se o número de 0's ou 1's é o máximo permitido, a ação é preencher a primeira posição vazia da linha com o valor que não atingiu lotação máxima
    b. Para cada coluna não preenchida: se o número de 0's ou 1's é o máximo permitido, a ação é preencher a primeira posição vazia da coluna com o valor que não atingiu lotação máxima;
    c. Para cada posição do *board* (da esquerda para a direita, de cima para baixo):
        i. se o conjunto dos valores dela própria mais as suas posições imediatamente adjacentes na horizontal tem dois valores repetidos e um nulo), a ação é colocar o valor não-presente na posição vazia
        ii. se o conjunto dos valores dela própria mais as suas posições imediatamente adjacentes na vertical tem dois valores repetidos e um nulo, a ação é colocar o valor não-presente na posição vazia
2. Ações "possíveis", mas não "certas" (da esquerda para a direita, de cima para baixo; relembrar que, para entrar nesta secção, é necessário que não existam ações "certas"):
    a. As duas ações são: preencher a primeira posição vazia com 0, preencher a primeira posição vazia com 1

Assim, o fator de ramificação máximo é 2.

Heurística

Uma heurística é um custo estimado do caminho até ao nó objetivo.

Como a cada aplicação de uma ação é colocado um valor, uma boa aproximação de distância ao nó objetivo será o número de posições por preencher. Contudo, como a cada iteração se preenche uma e uma só posição, e sempre pela mesma ordem, o valor de heurística será igual para cada nó – logo, para diminuir tempo de computação, "salta-se o intermediário" e devolve-se logo o valor 1 (foi testada uma outra heurística que realmente devolve o número de posições por preencher, e essa tem resultados iguais no que toca a número de nós gerados e expandidos, mas tempo de execução ligeiramente maior nos algoritmos de procura gananciosa e A*).

A heurística não é admissível exclusivamente pois h(n) <= h*(n) é falso para n=objetivo.

Resultados obtidos com as várias procuras

[Como cada nó tem o mesmo valor de heurística, cada nó terá o mesmo resultado na função de avaliação, pelo que os nós expandidos e gerados na procura gananciosa e A* são iguais, diferindo estas apenas no tempo de execução – ligeiramente maior em A*, por ter ainda de calcular a função de avaliação de cada nó.]

Seja *N* o número de posições inicialmente por preencher:

Procura	Completa?	Complexidade temporal	Complexidade espacial
largura primeiro	Sim. Expande todos os nós em cada iteração, e como não há ciclos infinitos, eventualmente um dos nós expandidos será o objetivo	O(2^*N*) (para ~0 ações certas) Geralmente 2^N pois expande todos os nós.	O(2^*N*) (para ~0 ações certas). Guarda todos os nós por expandir em memória, no pior caso.
profundidade primeiro	Sim. Como as *actions()* definem uma árvore quase-binária e sem ciclos, a profundidade máxima é igual a *N*, e esta procura é completa para profundidade máxima não-infinita	O(2^*N*) (para ~0 ações "certas" e primeira ação "possível mas não certa" sempre inversa à do caminho até ao objetivo) Geralmente menor que 2^N pois não expande todos os nós.	2**N* Apenas é guardado um caminho, e a profundidade do mesmo é sempre igual a *N*
gananciosa	Sim. Em cada iteração apenas uma posição é atualizada, e sempre pela mesma ordem relativa; como a heurística é igual para todos os nós, equivale a uma procura em largura primeiro (em termos de completude inclusive)	O(2^*N*) Como cada nó tem o mesmo valor de heurística, serão expandidos pela ordem de geração, sendo equivalente a PLP (mais o tempo de "calcular" a heurística)	O(2^N) Como cada nó tem o mesmo valor de heurística, serão expandidos pela ordem de geração, e os mesmos nós serão guardados que numa PLP
A*	Sim. Como explicado no aparte acima, neste caso em particular, apenas difere da procura gananciosa em termos de tempo	Semelhante a p. gananciosa mais o tempo de calcular f(n), como explicado no aparte acima	O(2^*N*) Como cada nó tem o mesmo valor de f(n), tem igual funcionamento espacial a p. gananciosa

Assim, como o pior caso (temporalmente) de cada uma das procuras é semelhante, o pior caso (espacialmente) da procura em profundidade primeiro é o melhor dos quatro, e o melhor caso da procura em profundidade primeiro (temporalmente) é melhor que o das outras pesquisas, a procura em profundidade primeiro é a melhor a utilizar neste problema.

Gráficos de Teste