Pesquisa aplicada ao Problema de Alocação de Lotes de Terreno

Relatório Final

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Inteligência Artificial

Grupo 10:
Paulo Bordalo Marcos - 201100759
Joao Manuel Ferreira Trindade - 201109221

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

30 de Maio de 2014

Resumo

Este trabalho consiste em implementar, utilizando algoritmos de pesquisa estudados nas aulas de Inteligência Artificial, uma resolução ao problemas de alocação de lotes de terreno cumprindo algumas regras estipuladas à priori devolvendo uma boa optimização do problema minimizando os custos.

Este problema irá ser tratado através de Pesquisas Sistemáticas/Informadas, como o Algoritmo A*, e Heuristícas associadas às caracteristicas do problema.

Para a implementação deste problema poderiamos utilizar uma destas duas linguagens: PROLOG ou JAVA. Foi escolhida a linguagem JAVA, pois para além de ser uma linguagem ao qual o grupo está melhor adaptado a mesma permite usar alguns tratamentos em grafos que já existem e simplificam a organização do projeto, sabendo de antemão que a linguagem PROLOG simplificaría vários outros aspectos deste problema.

1. Introdução

O principal foco deste projeto é a resolução de um problema de otimização, relativo a alocação de lotes de terreno, através da utilização do Algoritmo A* para a pesquisa de soluções dentro de um grafo de estados do problema.

2. O Problema

2.1 Descrição do Problema

Pretende-se atribuir lotes de terreno de uma cidade a obras/utilizações que devem obedecer a determinados critérios impostos pela utilização. Cada lote de terreno tem várias propriedades, entre elas: inclinação, dimensão, se tem estrada, se tem lago e se o solo é estável.

Ao criar uma utilização para um terreno o utilizador define quais são os critérios que esse deverá cumprir dentro do conjunto das restrições previamente descritas, estes critérios podem ser de cariz forte ou fraco. Um critério forte, é uma obrigatoriedade, o terreno solução, têm obrigatoriamente de seguir essa restrição. Um critério fraco não é obrigatorio, mas a sua validação é preferencial.

2.2 As Regras

A implementação e resolução do problema tem de obedecer a um número de regras e restrições. Estas são referentes a cada tipo de obra e a cada lote de terreno. Existem também regras principais que devem ser cumpridas independentemente das outras.

Existem dois tipos de regras , fortes e fracas:

Fortes - Restrições que devem ser cumpridas obrigatoriamente.

Fracas - Restrições que deve ser satisfeitas se for possível satisfaze-las após as restrições fortes já estarem todas implementadas.

Exemplo de regras possíveis das obras e terrenos:

1. Terrenos usados para Recreios devem estar preferencialmente localizados junto a um lago;

2. Evitar terrenos com inclinação para construção, cemitérios e lixeiras;

3. Solo pobre deve ser evitando para usos que envolvam construção;

4. A autoestrada é feia e barulhenta e deve ser evitada aquando da alocação de apartamentos, complexos residenciais ou áreas de recreio.

Regras principais:

- Um lote não pode ser utilizado para mais do que um uso;

- Todos os usos devem ser satisfeitos;

- A um uso apenas deve ser atribuído um lote;

- Minimizar o custo de aquisição dos terrenos;

3. Especificação

3.1 Representação de Conhecimento - Restrições

Neste ficheiro cada linha representa uma obra. A obra é definida na seguinte ordem:

<Nome> <Restrições>

As <Restrições> são o conjunto de restrições que essa obra necessita:

- Estabilidade do solo
- Acesso a Estrada
- Terreno conter um lago
- Grau de Inclinação do Terreno
- Dimensão

<SoloEstavel> <RSolo_Forte> <Estrada> <REstrada_Forte> <Lago> <RLago_Forte> <Inclinaçao> <RInclinaçao_Forte> <Dimensao> <RDimensao_Forte>

As tags em cima apresentadas representam:

<SoloEstavel> - Valores possiveis : {0,1} - Se 1 é necessário solo estável, se 0 não é necessário.

<Estrada> - Valores possiveis : {0,1} - Se 1 é necessário ter autoestrada, se 0 não é necessario

<Lago> - Valores possiveis : {0,1} - Se 1 é necessário lago, se 0 não é necessário.

<Inclinação> - Valores possiveis : {1,2,3} - Se 1 tem inclinação reduzida , se 2 tem inclinação normal, se 3 tem inclinação elevada.

<Dimensão> - Valores possiveis : {1,2,3} - Se 1 tem dimensão pequena , se 2 tem dimensão normal, se 3 tem dimensão grande.

Cada restrição tem associado um valor que determina se essa restrição é do tipo Forte - têm de ser cumprida, ou do tipo fraca - era conveniente que fosse cumprida, mas não e uma obrigação. Este valor é representado pelas Tags acima assinaladas a vermelho.

Esta representação permite que internamente guardemos todos os dados das restrições em variaveis do tipo inteiro, e se a restriçao é forte, ou fraca, numa variavel boleana.

```
1  habitacao 1 0 0 1 1 0 1
2  lixeira 0 0 0 1 1 0 0
3  recreio 0 0 1 0 0 0 0
4
```

Imagem 1: Ficheiro de Restrições

3.2 Representação do Conhecimento: Lotes de Terreno

Neste ficheiro cada linha representa um lote de terreno. O lote é definido na seguinte ordem:
<Características> As <Características> são o conjunto de características desse lote:

<Solo> <Inclinação> <Estrada> <Lago> <Dimensão> <Preço>

As tags em cima apresentadas representam:

<Solo> - Valores possiveis : {1,2} - Se 1 é solo estável, se 0 é instável.

<Inclinação> - Valores possiveis : {1,2,3} - Se 1 tem inclinação reduzida , se 2 tem inclinação normal, se 3 tem inclinação elevada.

<Estrada> - Valores possiveis : {0,1} - Se 0 não tem auto estrada , se 1 tem autoestrada.

<Lago> - Valores possiveis : {0,1} - Se 1 tem lago, se 0 tem.

<Dimensão> - Valores possiveis : {1,2,3} - Se 1 tem dimensão pequena , se 2 tem dimensão normal, se 3 tem dimensão grande.

<Preço> - Valores possiveis : [0.0,∞] - Representa o preço do lote.

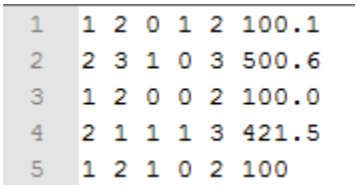

Imagem 2: Ficheiro de Lotes de Terreno

3.2 Algoritmo A*

O algoritmo A* é uma mistura do Custo Uniforme com métodos gananciosos, no sentido em que:

Custo uniforme: Apenas tem em conta os custos até ao sucessor

$$C(S; S0) = c(S) + c(S0).$$

Métodos gananciosos: Apenas tem em conta a heurística do sucessor

$$H(S; S0) = h(S0).$$

A*: O valor de cada sucessor é dado por

$$f(S; S0) = C(S; S0) + H(S; S0).$$

Ou seja, se todos os nós tiverem custo 0, o algoritmo passa a ser uma pesquisa gananciosa e se a heurística for constante, o algoritmo é igual ao Custo Uniforme. Se todos os nós tiverem custo constante e a heurística for constante, este algoritmo passa a ser uma Pesquisa em Largura.

Esta algoritmo utiliza duas listas:

Lista aberta: Nós a expandir (semelhante ás las da Pesquisa em Largura e do Custo Uniforme). Deve ser uma la de prioridades ordenada por f;

Lista fechada: Nós já expandidos (pode ser representada implicitamente como um estado do nó).

O algoritmo funciona da seguinte forma (tendo uma lista aberta apenas com o estado inicial e uma lista fechada vazia):

1. Se a lista aberta estiver vazia, o algoritmo termina;

2. Retira-se o primeiro estado da lista aberta e adiciona-se-o à lista fechada (ou marca-se como expandido);

3. Se o estado for o estado nal, o algoritmo retorna;

4. Caso contrário, para cada sucessor que não esteja na lista fechada:

   (a) Calcula-se o seu possível novo valor estimando, f0(S0) = f(S; S0)

   (b) Se o sucessor não estiver na lista aberta ou f0(S0) (novo valor estimado para S') for melhor que f(S0) (valor estimado para S' anterior), o sucessor é adicionado (ou actualizado) na lista aberta.

5. Repete-se o algoritmo.

Análise do Algoritmo

A complexidade do algoritmo A* depende muito da heurística, no entanto, para todos os casos, este algoritmo é bastante dispendioso na memória, visto que tem de armazenar informação de todos os estados até chegar ao estado final. Como tal, em problemas de larga escala são usadas outras variantes deste algoritmo como o IDA* (Baseado no Aprofundamento Progressivo, mas com custo máximo em vez de profundidade limite).

3.3 Heurística

Uma boa heurística deve sub-estimar a distância de um estado à solução, sendo tanto melhor quanto mais se aproximar da distância real entre esse estado e a solução.

A heurística utilizada por nós é baseada na apresentada pelo autor Ram D. Sriram no livro intitulado “Inteligent Systems for Engineering: A knowledge-based approach”.

Para avaliar a distância heurística de um estado à solução, de forma a minimizar o custo da solução, tem-se em conta o custo dos terrenos mais baratos ainda possíveis de serem alocados, por exemplo:

Se num estado A, faltarem alocar 3 terrenos, a distância heurística do estado A à solução, para efeitos de processamento pelo algoritmo A*, será a soma do custo dos 3 terrenos mais baratos que ainda não tenham sido alocados. No caso óptimo, esses três terrenos corresponderão ás 3 utilizações em falta, mas caso um desses terrenos não cumpra as restrições necessárias, então o custo total será sempre superior ao inicialmente estimado.

Além disto devemos ainda ter em conta, no calculo heuristico o numero de restrições fracas cumpridas por uma possivel solução.

Consideramos que a nossa heuristica deve sempre preferir um terreno com uma restrição fraca cumprida, desde que este não custe mais do dobro do terreno semelhante, sem a restrição fraca.

A formula final do calculo da heuristica, para um estado em que faltem alocar terrenos para N localizações é:

$$\frac{somatorioLotesMaisBaratos(N)}{1 + numeroRestricoesFracasCumpridas}$$

em que "somatorioLotesMaisBaratos(N)" é a soma dos custos dos N lotes mais baratos, do conjunto de lotes ainda não alocados, e o "numeroRestricoesFracasCumpridas" é o numero total de restrições fracas cumpridas por o estado atual.

3.4 Cenários de Teste

Um exemplo de um resultado esperado, pode ser visto através do seguinte caso, os ficheiros de teste utilizados foram, com os lotes de terreno "cidade4.txt", e com as restricoes "teste4-1.txt" :

Lotes da cidade:

Lote 0 : Solo Estavel | Inclinacao Normal | Sem Estrada | Sem Lago | Dimensao Normal | 1000.1€
Lote 1 : Solo Estavel | Inclinacao Elevada | Com Estrada | Sem Lago | Dimensao Grande | 500.2€
Lote 2 : Solo Instavel | Inclinacao Normal | Com Estrada | Com Lago | Dimensao Normal | 650.1€
Lote 3 : Solo Instavel | Inclinacao Normal | Sem Estrada | Com Lago | Dimensao Grande | 1200.4€
Lote 4 : Solo Estavel | Inclinacao Elevada | Sem Estrada | Sem Lago | Dimensao Pequena | 985.5€

Obras que se pretende efectuar na cidade:

Habitação 1 - Restrições:
- Tem de ter Solo Estável.

Lixeira 1 - Restrições:
- Tem de ter Estrada.
- Tem de ter Inclinação Normal.

Parque 1 - Restrições:
- Tem de ter Lago.

Hospital 1 - Restrições:
- Tem de ter Estrada.
- Tem de ser Dimensão Grande.

Com estas restrições e características, os seguintes lotes são possíves de ser atribuídos da seguinte forma:

Habitação 1 - Pode ser construido nos lotes 0, 1 e 4.

Lixeira 1 - Pode ser construido no lote 2.

Parque 1 - Pode ser construido nos lotes 2 e 3.

Hospital 1 - Pode ser construido no lote 1.

Assim sendo a resolução passa por alocar da seguinte forma:

Habitação 1 - Lote 4.

Lixeira 1 - Lote 2.

Parque 1 - Lote 3.

Hospital 1 - Lote 1.

Custo Total : 3336.2€

O Parque 1 não fica no Lote 4 pois como tinha o Lote 3 e 4 disponíveis mas a Lixeira 1 tinha apenas possibilidade no Lote 3, ficou assim atribuido de forma a que os dois pudessem ter um Lote livre.

Resultado Obtido:

```
Resultado da Alocação de Terrenos:
>Habitacao1 -> Lote nº: 4 > Custo: 985.5
>Lixeira1 -> Lote nº: 2 > Custo: 650.1
>Parque1 -> Lote nº: 3 > Custo: 1200.4
>Hospital1 -> Lote nº: 1 > Custo: 500.2

Custo total: 3336.2 Numero de Restricoes Fracas Cumpridas: 0
```

Imagem 3: Resultado de Alocação

4 Desenvolvimento

4.1 Ferramentas de Desenvolvimento

Quanto às ferramentas de desenvolvimento utilizadas, como já referimos anteriormente, optamos por usar o Java como linguagem de programação, os principais motivos para esta escolha foram o grande avontade dos membros do grupo com o paradigma da programaçao orientada a objectos uma vez que este tem sido o mais utilizado nos ultimos anos por outras unidades curriculares, bem como a versatilidade e facilidade da programação em Java. Foi usada a verão mais recente do Java

O ambiente de desenvolvimento escolhido foi o Eclipse IDE, na sua mais recente versão.

4.2 Interface

Para se introduzir as obras que se pretende efectuar e os lotes disponíveis, é possível faze-lo de duas formas: inserir manualmente ou carregar um ficheiro de texto.

O programa tem um menu em modo consola que permite adicionar manualmente ou escolher a opção de carregamento.

```
1 - Carregar Terreno
2 - Carregar Restricoes
3 - Nova Restricoes
4 - Atribuir Terrenos -> Utilizacoes
5 - Sair
```

Imagem 4 - Interace do Utilizador

4.3 Estruturação da Aplicação

A aplicação está estruturada em quatro simples classes, a classe main, com os menus e interface de utilizador, bem como a implementação do algoritmo A-Star. A classe Lote representa cada lote de terreno e as suas caracteristicas. A classe Utilização representa cada Utilização para a qual queremos alocar um terreno, bem como as restrições da mesma. Por último a classe estado, é a classe utilizada na implementação do algoritmo A-Star, cada objecto desta classe é um estado intermédio na pesquisa da solução. O estado inicial, é, por exemplo, um estado em que nenhuma utilização tem um lote de terreno alocado.

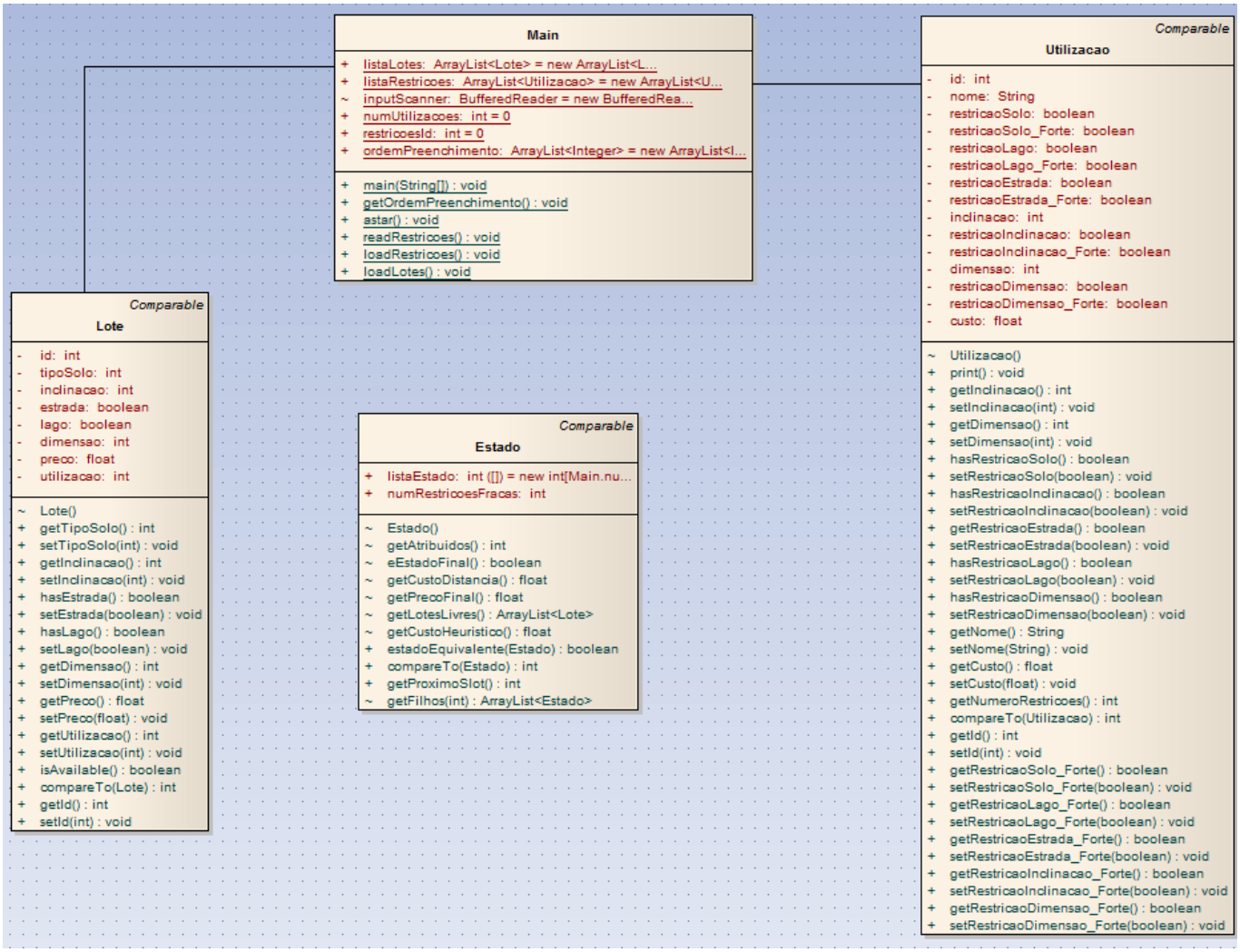

Imagem 5: Diagrama de Classes

4.4 Recursos Externos

Na relatório intermédio deste trabalho, tinhamos indicado que iriamos utilizar a biblioteca externa JGraphT para a representação de grafos, nomeadamente, para o grafo de estados produzido na execução do Algoritmo A-Star, contudo, a utilização destas ferramentas provou-se desnecessária, acabando por serem descartadas da solução hoje apresentada pela grupo.
Este facto deveu-se a algo que só constatamos durante a implementação do acima referido algoritmo de pesquisa. Uma vez que cada "estado" da nossa implementação ocorre apenas uma vez, ou seja, cada nó "estado" tem apenas um "nó-pai", o nosso grafo de estados, é na verdade uma arvore. Isto acontece porque cada "estado" guarda em si próprio as decisões que ja foram tomadas anteriormente, podendo apenas gerar "nós-estado-filhos" de decisões de alocação que ainda não foram tomadas. Este facto, apesar de algo complicado de explicar, é facilmente entendido observando a imagem de seguida, retirada também de [RAM D. SRIRAM - "Inteligent Systems for Engineering: A knowledge-based approach"]

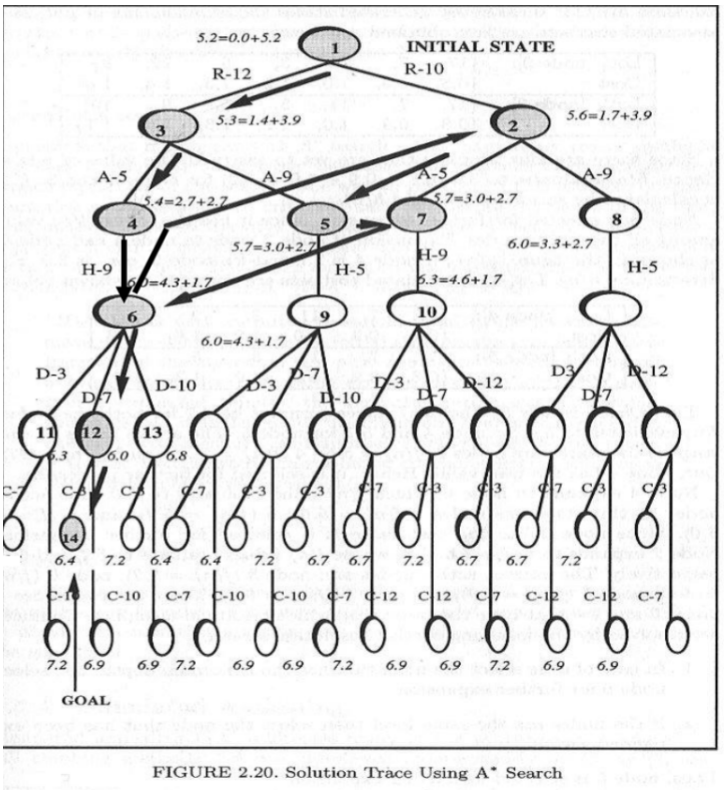

FIGURE 2.20. Solution Trace Using A* Search

Imagem 6 - Árvore de Pesquisa

Uma vez que cada nó só pode ter um pai, após esse nó ter sido expandido, e colocado na lista fechada, ele nunca mais é utilizado, como não haverá outro caminho para esse nó, nunca será necessário re-calcular distancias entre nós. Isso permite uma implemtentação direta e simplista do algoritmo A*, em que apenas é necessario utilizar duas listas (Lista aberta e Lista Fechada) para a organização dos dados, dispensando assim a utilização de um grafo, e poupando recursos computacionais.

6. Conclusões e Melhorias Futuras

Em suma, este projecto fui muito util para por em pratica o desenvolvimento, de um algoritmo de pesquisa informada, algo que ainda não havia sido explorado em anteriores unidades curriculares que versavam tambem sobre a analise de algoritmos de pesquisa.
Futuramente gostariamos de implementar um módulo que permitisse graficamente visualizar a alocação dos terrenos, bem como analisar graficamente a evolução do algoritmo, desde o seu estado inicial ao estado final.

7. Recursos

Bibliografia

[1] Resumo Conjunto Online - Inteligência Artificial - Estudantes do Mestrado Integrado Engenharia Informática e Computação da FEUP.

[2] Ram D. Sriram - Livro - "Inteligent Systems for Engineering: A knowledge-based approach". - Springer - http://www.amazon.com/Intelligent-Systems-Engineering-Knowledge-based-Approach/dp/3540761284#reader_3540761284

[3] Eugénio Oliveira - "Métodos de Resolução de Problemas e Algoritmos para a Evolução" - Apontamentos das aulas teóricas - Inteligência Artificial - MIEIC FEUP - http://www.fe.up.pt/~eol/IA/ia1213.html

Elementos do grupo

Paulo Bordalo Marcos - 50 % do trabalho prático
Joao Manuel Ferreira Trindade - 50 % do trabalho prático