Universidade do Porto

Faculdade de Engenharia

FEUP

Inteligência Artifical

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Pesquisa aplicada à gestão de projetos

Relatório Intercalar

Diogo Afonso Duarte Reis up201505472@fe.up.pt
Francisca Leão Cerquinho Ribeiro da Fonseca up201505791@fe.up.pt
Mariana Lopes da Silva up201506197@fe.up.pt

Docentes:
Prof. Ana Paula da Cunha Rocha
Prof. Eugénio Oliveira
Prof. Henrique Daniel de Avelar Lopes Cardoso

8 de Abril, 2018

Contents

1 Objetivo

No âmbito da unidade curricular Inteligência Artificial foi-nos proposto a realização de um projeto que aborda a pesquisa sistemática/informada de soluções e estudo da complexidade teórica e empírica dos algoritmos implementados, utilizando a linguagem de programação Java. O presente trabalho tem por objetivo determinar a melhor alocação de elementos às várias tarefas de um projeto, maximizando o desempenho global, através da escolha dos elementos com o melhor desempenho possível para determinada competência.

A notar que, as tarefas podem ter precedências entre si e uma duração associada (pessoa/mês). Cada elemento candidato possui um conjunto de competências e respetivos desempenhos, que permitem satisfazer uma ou mais tarefas. Um elemento não pode estar associado a mais que uma tarefa ao mesmo tempo e o seu desempenho varia entre 1 a 5, sendo que associamos 1 ao melhor desempenho e 5 ao pior. A escala foi invertida devido à utilização do algoritmo A*, como método de pesquisa, que procura um mínimo como solução final.

2 Descrição

2.1 Especificação

Para este problema de alocação de recursos, considera-se que a ordem de execução das tarefas faz parte dos dados de entrada do problema. Dessa forma, uma vez determinada, previamente, a sequência de execução das tarefas, o presente problema resume-se em pesquisar e definir quais os elementos usados para executar as tarefas, de modo a maximizar o desempenho. Assume-se que, o elemento alocado numa tarefa trabalha nela desde o início até à sua conclusão.

2.1.1 Representação do tema como problema de pesquisa: estados, função de transição, heurística

De modo a responder ao problema inicial, concebemos um modelo que recebe como *input* dois ficheiros de texto, nomeadamente, o *elements.txt* que guarda a informação de cada elemento (nome, competências e respetivos desempenhos) e o *tasks.txt*, que guarda a informação das diferentes tarefas (identificador, duração, competência requerida e lista de precedências).

Para isso, foi decidido usar um grafo ***G(N,A)*** com um conjunto de nós, ***N***, e um conjunto de arestas, ***A***. O estado de cada nó do grafo engloba três estruturas de dados, uma que representa as tarefas alocadas (*HashMap<Integer, String> allocatedResources*), isto é, cada tarefa tem associado um elemento, outra que representa a lista de tarefas ainda por alocar (*ArrayList<Integer> tasksToAllocate*) e outra que representa os elementos que ainda não foram atribuídos (*ArrayList<Integer> unassignedElements*). Tal como foi referido anteriormente, é fornecida uma lista de precedências para cada tarefa que será a função de transição dos estados, determinando, assim, a ordem de execução das tarefas.

A heurística a aplicar para esta problemática é baseada numa estimativa do desempenho dos indivíduos que se consegue a partir do nó atual até ao final do projeto, utilizando como método de pesquisa o algoritmo A*, e para efeitos de comparação de resultados, o algoritmo Guloso.

2.1.2 Algoritmos de pesquisa a aplicar

Algoritmo A*

O algoritmo A* é um algoritmo de pesquisa informado, que escolhe o "melhor primeiro", fazendo uma pesquisa em Grafo de todos os caminhos possíveis para a solução, o de menor custo e entre esses caminhos, primeiro considera os que parecem levar mais rapidamente à solução. De um modo geral, A* seleciona o caminho que minimiza:

F(n) = G(n) + H(n), onde ***n*** é o nó corrente, ***G(n)*** é o custo do caminho do nó inicial para n, e ***H(n)*** é uma heurística que representa a estimativa do custo do passo, desde n até à solução. A heurística deve tentar ser mais simples que o cálculo do valor real do estado.

Como *F(n)* depende de *G(n)*, em cada nível não se escolhe só "o mais promissor", que corresponde à melhor estimativa para o futuro, mas a melhor combinação da promessa com o melhor passo até então, isto é, o menor custo até ao momento.

No caso concreto do problema de alocação de recursos, o valor de G é o desempenho dos elementos na execução das tarefas e o valor de H (heurística), como foi referenciado acima, é a estimativa do desempenho que se consegue desde n até ao final do projeto, sendo que H deve subestimar o valor real, para encontrar um caminho de custo ótimo.

Sendo o algoritmo A*, um algoritmo de minimização e o problema em questão um problema de maximização, consideremos que o desempenho de cada elemento numa dada competência é avaliado de 1 a 5, sendo 1 o desempenho máximo.

Imaginemos a seguinte situação:

Tarefas	Nr de meses necessários	Nr de pessoas necessárias	Competências necessárias	Precedentes
T1	1	2	java	[ ]
T2	2	1	python	[T1]
T3	1	1	web	[T1]
T4	2	1	c++	[T2,T3]

Tabela 1: Tarefas de um determinado projeto

Elemento	Competências	Desempenho
E1	[java, web]	[1,1]
E2	web	3
E3	[c++, python]	[4,1]
E4	python	2
E5	java	5

Tabela 2: Elementos disponíveis para atribuição

Representação formal:

Soli: Lista das atribuições de tarefas a elementos no estado i
Lti: Lista das tarefas ainda não atribuídas
Lei: Lista dos elementos por considerar

Estado inicial: Sol0=[ ], Lti=[T1,T2,T3,T4], Lei=[E1,E2,E3,E4,E5]

Objetivo: Soli = [T1-E5,T2-E4,T3-E2,T4-E3] Lti = [ ] Lei = [E1]

Função heurística: F(n) = G(n) + H(n)
H(n): soma do custo dos elementos p, com pior desempenho, ainda não atribuídos;
p: é o número de elementos ainda não contemplados;
G(n): custo da solução parcial até ao momento.

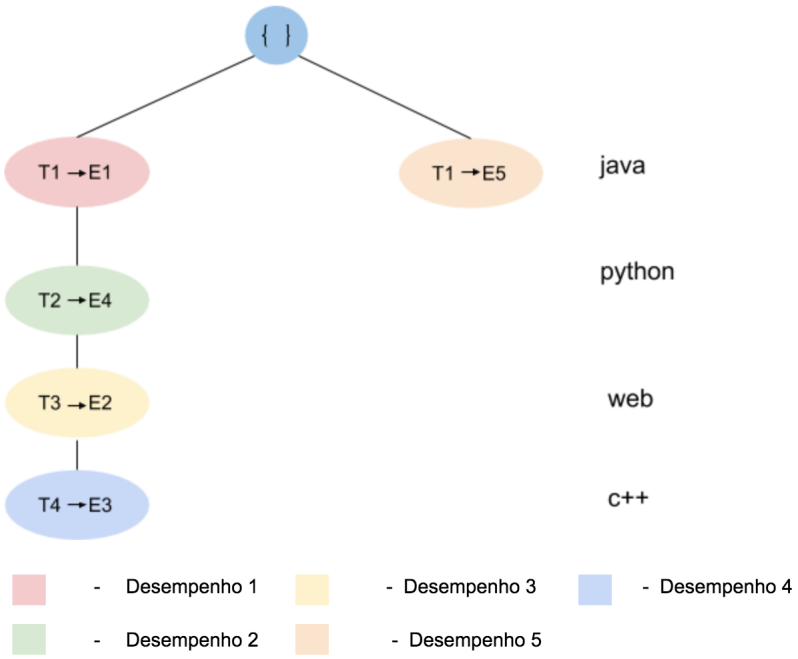

Figura 1: Grafo do exemplo apresentado

Algoritmo Guloso

O algoritmo heurístico guloso (*greedy*), também denominado por método ganancioso, resolve problemas de otimização (maximização ou minimização), selecionando a cada iteração a escolha que melhor atende a função objetivo do problema naquele exato momento, isto é, escolhe, deterministicamente, uma solução que representa um ótimo local, sem considerar as consequências futuras, julgando, assim, que através desta estratégia de pesquisa encontra a solução ótima global para o problema.

2.2 Trabalho efetuado

Após a análise detalhada do problema, o grupo começou por planear uma proposta de solução. Primeiramente, tratou-se dos dados de entrada do problema e para isso foram criados dois ficheiros de texto, *elements.txt* e *tasks.txt*, em que o conteúdo de

ambos se encontra separado por ponto e vírgula e em caso de múltiplos itens do mesmo tipo, escreve-se essa informação como um *array*, isto é, elementos separados por uma vírgula caso o array tenha mais que um elemento e delimitados por um parênteses reto à esquerda e à direita, tal como se encontra no exemplo seguinte:

elements.txt:
Rui;[java,c];[5,3]
Joao;[web,java];[3,3]

tasks.txt:
1;2;1;java;[]
2;1;1;java;[1] 3;1;1;web;[1,2]

Para extrair a informação, lê-se conteúdo dos ficheiros linha a linha, ponto e vírgula a ponto e vírgula, preenchendo os campos das classes previamente criadas, *Element.java* e *Task.java*.

Posteriormente, definiu-se os estados de cada nó, a função de transição e as heurísticas que serão utilizadas nos métodos de pesquisa a aplicar, o algoritmo A* e o algoritmo Guloso.

2.3 Resultados esperados e forma de avaliação

De forma a validar o resultado do trabalho, serão testados vários tipos de conjeturas que podem ocorrer num problema de alocação de recursos a tarefas, para diferentes algoritmos.

Efetivamente, os testes passarão por verificar e analisar os resultados obtidos, para cada um dos algoritmos implementados, A* e Gulosa, nas seguintes situações:

1. Muitas tarefas e poucos recursos;
2. Poucas tarefas e muitos recursos;
3. Número de tarefas e recursos igual.

A análise recairá sobre qual dos algoritmos devolve o melhor desempenho das tarefas. O algoritmo A* é uma mistura do algoritmo de Custo Uniforme com métodos gananciosos, no sentido em que:

- O algoritmo de Custo uniforme: apenas tem em conta os custos até ao sucessor, *G(n)*. Este método realiza uma pesquisa ótima e completa, mas por ser não informado utiliza uma estratégia baseada em tentativas de solução por força bruta, o que pode gerar um alto custo computacional. Este algoritmo calcula o caminho de custo mínimo entre um nó de origem e um nó de destino. Para isso, ele utiliza a função de avaliação de custo de caminho: *F(n) = G(n)*, sendo *G(n)* o custo do caminho já percorrido partindo da raiz até o nó n.

- Algoritmo Guloso (ganancioso): apenas tem em conta a heurística do sucessor, *H(n)*. Este método realiza uma pesquisa informada, que procura minimizar os custos computacionais empregando, além da definição do problema, conhecimento específico acerca do problema em questão. Este algoritmo não é ótimo nem completo, mas com uma boa heurística pode encontrar mais rapidamente uma solução, gerando um menor custo computacional.

- A*: o valor de cada sucessor é dado por *F(n)=G(n)+H(n)*. Este método combina as duas estratégias anteriormente referidas, por isso, é ótimo, completo e, com uma boa heurística, pode minimizar significativamente o custo computacional da pesquisa, ou seja, a complexidade de tempo e de espaço.

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. 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. Por outro lado, A* é ótimo quando a heurística é consistente. Já o algoritmo Guloso, apesar de simples e de fácil implementação, nem sempre conduz a soluções ótimas globais e pode efetuar cálculos repetitivos.

3 Conclusões

A realização desta fase inicial do projeto, para além de nos ser muito útil do ponto de vista algorítmico e estrutural, permitiu-nos perceber os desafios presentes na implementação e estruturação dos métodos de pesquisa A* e a Gulosa, que requerem um estudo complexo, mesmo antes de serem postos em prática. Assim, foi possível consolidar os nossos conhecimentos e capacidade de análise em relação a problemas de pesquisa de soluções.

Em suma, com este relatório intercalar fez-se um planeamento objetivo do que se terá de desenvolver posteriormente, através do debate, do grupo com a docente, de estratégias de implementação dos algoritmos no contexto de alocação de tarefas a recursos.

4 Recursos

[**1**] A* Search Algorithm. Acedido em 2018. Disponível em https://en.wikipedia.org/wiki/A*_search_algorithm.
[**2**] Heuristics. Acedido em 2018. Disponível em http://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html.
[**3**] Greedy algorithm. Acedido em 2018. Disponível em https://en.wikipedia.org/wiki/Greedy_algorithm.