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 Final

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

20 de Maio, 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. 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 Especificação

2.1 Análise detalhada do tema

Para este problema da alocação de recursos a tarefas, considera-se que a ordem de execução destas faz parte dos dados de entrada do problema, através da listagem de precedências de cada uma das tarefas. Dessa forma, uma vez determinada, a sequência de execução das tarefas, o presente problema resume-se em pesquisar e definir quais os elementos, com aptidões e desempenhos associados, que podem ser usados para executar as tarefas com competências específicas, de modo a maximizar o desempenho. Assume-se que o elemento alocado numa tarefa trabalha nela desde o início até à sua conclusão. Assume-se que cada tarefa tem apenas uma competência necessária e que as tarefas têm, invariavelmente, como duração um mês, variando, apenas, o número de pessoas necessárias para a sua realização. A acrescentar que, cada tarefa pode ou não ter precedências. Caso tenha mais do que uma, é apenas necessário que uma delas se realize antes da execução da mesma.

2.1.1 Ilustração de cenários

Existem vários cenários possíveis para esta problemática de alocação de recursos a tarefas, nomeadamente quando existem muitas tarefas com várias competências requeridas e poucos recursos disponíveis, poucas tarefas e muitos recursos, o número de tarefas e recursos é igual ou ainda, quando existem tarefas em paralelo. Imaginemos, então, as seguintes situações:

Situação 1: (muitas tarefas e poucos recursos)

Tarefas	Nº de pessoas necessárias	Competências necessárias	Precedências
T1	1	Web	[ ]
T2	1	Java	[T1]
T3	2	Phython	[T2,T3]
T4	1	C	[T3]
T5	1	Java	[T4]

Tabela 1: Tarefas de um determinado projeto

Elemento	Competências	Desempenho
João	[Web, C, Python]	[3,3,2]
Rui	[Java, Phython]	[5,3]

Tabela 2: Elementos disponíveis para atribuição

Resultado Esperado

T1	João
T2	Rui
T3	[João,Rui]
T4	João
T5	Rui

Tabela 3: Resultado da situação 1

Situação 2: (poucas tarefas e muitos recursos)

Tarefas	Nº de pessoas necessárias	Competências necessárias	Precedências
T1	1	Web	[ ]
T2	1	Java	[T1]

Tabela 4: Tarefas de um determinado projeto

Elemento	Competências	Desempenho
João	[Web, C]	[3,3]
Rui	[Java, Phython]	[5,3]
Maria	[Java]	[4]
Joana	[C]	[4]
Rita	[Phython, C]	[2,1]
Pedro	[Web]	[1]

Tabela 5: Elementos disponíveis para atribuição

Resultado Esperado

T1	Pedro
T2	Maria

Tabela 6: Resultado da situação 2

Situação 3: (número de recursos e tarefas igual)

Tarefas	Nº de pessoas necessárias	Competências necessárias	Precedências
T1	1	Web	[ ]
T2	3	Java	[T1]
T3	2	Phython	[T2,T3]
T4	1	C	[T3]

Tabela 7: Tarefas de um determinado projeto

Elemento	Competências	Desempenho
João	[Web, Java]	[3,3]
Rui	[Java, C]	[5,3]
Maria	[Java, Phython]	[2,1]
Rui	[Java, Web, Phython]	[4,3,2]

Tabela 8: Elementos disponíveis para atribuição

Resultado Esperado

T1	João
T2	[João,Maria,Pedro]
T3	[Maria,Pedro]
T4	Rui

Tabela 9: Resultado da situação 3

Situação 4: (tarefas em paralelo)

Tarefas	Nº de pessoas necessárias	Competências necessárias	Precedências
T1	1	Web	[ ]
T2	1	Java	[T1]
T3	2	Java	[T1]
T4	1	C	[T2,T3]
T5	2	Web	[T4]

Tabela 10: Tarefas de um determinado projeto

Elemento	Competências	Desempenho
João	[Web, Java]	[3,3]
Rui	[Java, C]	[5,3]
Maria	[Java]	[2]
Pedro	[Java, Web]	[4,3]

Tabela 11: Elementos disponíveis para atribuição

Resultado Esperado

T1	João
T2	Maria
T3	Rui
T4	[João, Pedro]

Tabela 12: Resultado da situação 4

A notar que, para que alocação de recursos a tarefas se realize é necessário haver recursos suficientes com competências para as tarefas requeridas, caso contrário, o nosso programa alerta o utilizador dessa ocorrência.

2.2 Abordagem

2.2.1 Algoritmos e sua breve explicação

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. Este algoritmo baseia-se na melhor combinação da promessa com o melhor passo até então, isto é, o menor custo até ao momento, uma vez que, 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.

Pesquisa em Largura

A Pesquisa em Largura é uma estratégia simples na qual o nó raiz é o primeiro a ser expandido, depois todos os sucessores do nó raiz e assim sucessivamente até que o melhor caminho possível seja encontrado. O algoritmo Pesquisa em Largura usa a fila como estrutura de dados. Esta estrutura é do tipo *FIFO (first-in first-out)*, onde o primeiro elemento a ser inserido, será o primeiro a ser retirado. Esta fila armazena todos os nós a explorar e cada vez que um nó é explorado é adicionado ao conjunto de nós visitados.

Se realizarmos uma pesquisa em largura num grafo, fará o seguinte:

1. Definir o nó 1 como o nó inicial
2. Adiciona este nó à fila
3. Adicionar este nó ao conjunto visitado
4. Se esse nó for o nosso nó final, retorna "true", caso contrário, inclui o Nó 2 e o Nó 3 na fila.
5. Verifica se o Nó 2 está na fila e, se não estiver, adiciona o Nó 4 e o Nó 5 à fila.
6. Analisa o próximo nó da fila, que deve ser o nó 3 e verifica o mesmo.
7. Se o Nó 3 não for o nosso nó final, adiciona o Nó 6 e o Nó 7 à fila.
8. Repete até que o Nó final seja encontrado.

Se parássemos a execução depois de ser verificado o Nó 3, a fila ficaria assim: Nó 4, Nó 5, Nó 7, Nó 8.

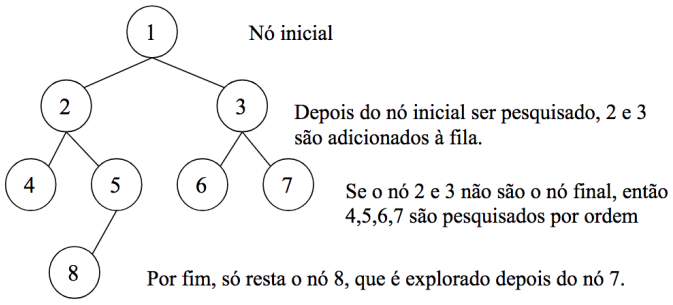

Figura 1: Exemplo de pesquisa em largura num grafo

Algoritmo de Custo Uniforme

O algoritmo de Custo Uniforme apenas tem em conta os custos até ao sucessor, $\boldsymbol{G(n)}$. 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: $\boldsymbol{F(n) = G(n)}$, sendo $\boldsymbol{G(n)}$ o custo do caminho já percorrido partindo da raiz até o nó n.

Algoritmo Guloso

O algoritmo Guloso 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.

O algoritmo 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.

2.2.2 Métricas e Heurísticas

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) 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	Nº de pessoas necessárias	Competências necessárias	Precedentes
T1	1	Web	[ ]
T2	1	Java	[ ]
T3	2	Phython	[T2]
T4	2	C	[T3,T1]
T5	1	Java	[T4]

Tabela 13: Tarefas de um determinado projeto

Elemento	Competências	Desempenho
João	[Web, C]	[3,3]
Rui	[Java, Phython]	[5,3]
Maria	[C]	[2]
Pedro	[Java,Web]	[4,3]
Mariana	[Phython]	5

Tabela 14: Elementos disponíveis para atribuição

Representação formal:

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

Estado inicial:
Sol0=[ ], Lti=[T0,T1,T4,T5]

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

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

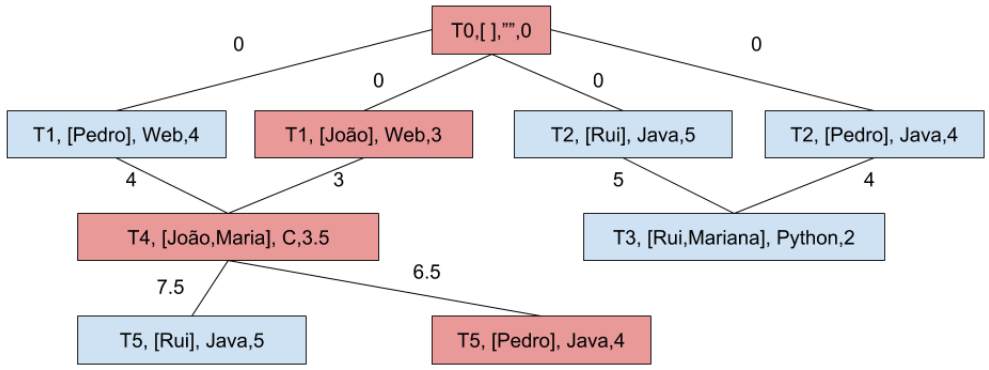

Figura 2: Grafo do exemplo apresentado com a utilização do algoritmo A*

3 Desenvolvimento

3.1 Linguagem de programção e ambiente de desenvolvimento

O projeto foi desenvolvido no sistema operativo Ubuntu, versão 17.04, utilizando o Eclipse como IDE. Java foi a linguagem de programação escolhida para a implementação.

3.2 Estrutura da aplicação

3.2.1 Detalhes relevantes da implementação

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 respectivos desempenhos) e o *tasks.txt*, que guarda a informação das diferentes tarefas (identificador, número de pessoas, competência requerida e lista de precedências). O conteúdo de ambos encontra-se separado por ponto e vírgula. Para extrair a informação, lê-se o 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*, que correspondem ao conjunto de recursos e tarefas, respetivamente. Para isso, foi decidido usar um grafo G(N,A) com um conjunto de nós, N, e um conjunto de arestas, A.

Tendo em conta o número de pessoas necessárias para cada realizar cada tarefa, o programa começa por guardar num array todas as combinações possíveis de elementos que têm a competência necessária para realizar a tarefa. Para cada combinação calcula a média do desempenho de cada grupo. Esta informação é guardada em cada nó do grafo da seguinte forma:

Nó(*String* Identificador da tarefa, *Array<String>* elementos, *String* competência necessária, *Integer* média do desempenho do grupo de elementos)

Aresta(*Nó* nó de Destino, *Double* desempenho)

Em seguida, são construídas as adjacências de cada nó. 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.

No final é invocado o algoritmo (A*, Gulosa, Custo Uniforme ou Pesquisa em Largura) com o nó inicial e final do grafo. O nó inicial é aquele que não tem precedências, e o nó final é aquele que não tem adjacências.

4 Experiências

De forma a validar e testar a implementação dos algoritmos mencionados anteriormente foram realizados vários casos de uso e de teste. Efetivamente, os testes passarão por verificar e analisar os resultados obtidos, para cada um dos algoritmos implementados, A*, Gulosa, Custo Uniforme e Pesquisa em Largura

Teste nº 1:

Tarefas	Nº de pessoas necessárias	Competências necessárias	Precedências
T1	1	web	[]
T2	1	java	[T1]
T3	2	java	[T1]
T4	1	c	[T2,T3]
T5	2	web	[T4]

Tabela 15: Tarefas do projeto de teste nº1

Elemento	Competências	Desempenho
Joao	[web,java]	[3,3]
Rui	[java,c]	[5,3]
Maria	[java]	[2]
Pedro	[java,web]	[4,3]

Tabela 16: Elementos disponíveis para atribuição nos 4 testes

Tarefa	Elemento
T1	João
T2	Maria
T4	Rui
T5	João, Pedro

Tabela 17: Resultado da alocação em A*, Custo Uniforme e Gulosa

Tarefa	Elemento
T1	Pedro
T3	Maria, Pedro
T4	Rui
T5	João, Pedro

Tabela 18: Resultado da alocação em Pesquisa em Largura

Os 4 algoritmos chegam a uma solução ótima tendo em conta a estrutura de cada um. Observa-se que, todos os algoritmos menos a Pesquisa em Largura obtêm o mesmo caminho percorrido. Isto deve-se ao facto de os 3 primeiros algoritmos percorrerem o caminho que calcularam ser mais curto, tendo em conta as tarefa 2 e 3 estarem em paralelo e a tarefa 3 ter um número de elementos necessários superior aos da tarefa 2.

O algoritmo de Pesquisa em Largura resulta numa solução diferente, isto é, escolhe a tarefa 3 em vez da 2, porque a tarefa 3 foi visitada (ou seja, entrou para a fila) antes da tarefa 2.

Em relação à alocação de elementos às tarefas repete-se a solução novamente entre os 3 primeiros algoritmos. A tarefa 1 requer especialidade em *web*, portanto tem de escolher entre o João e o Pedro que têm essa competência, sendo alocado o primeiro a ser visitado no grafo, já que o desempenho dos dois é igual com valor de 3.

Na tarefa 2 é escolhida a Maria, devido ao sua maior desempenho em Java. Na tarefa 3 são escolhidos dois elementos já que esta requer 2 pessoas. Este elementos são a Maria e o Pedro, devido às suas habilidades em Java. Na tarefa 4 é alocado o único elemento com competência em C, o Rui. Na tarefa 5 é alocado o João da mesma forma que na tarefa 1.

Teste nº 2:

Tarefas	Nº de pessoas necessárias	Competências necessárias	Precedências
T1	1	web	[]
T2	1	java	[T1]
T3	2	java	[T1]
T4	1	c	[T1]
T5	2	web	[T2,T3,T4]
T6	1	c	[T2,T3,T4]
T7	1	java	[T5,T6]

Tabela 19: Tarefas do projeto de teste nº2

Tarefa	Elemento
T1	João
T2	Maria
T5	João, Pedro
T7	Pedro

Tabela 20: Resultado da alocação em A* e Custo Uniforme

Tarefa	Elemento
T1	João
T2	Maria
T6	Rui
T7	Pedro

Tabela 21: Resultado da alocação em Gulosa

Tarefa	Elemento
T1	Pedro
T4	Rui
T6	Rui
T7	Pedro

Tabela 22: Resultado da alocação em Pesquisa em Largura

Neste teste, observa-se que os resultados do A* e Custo Uniforme são iguais, mas diferentes do resultados nos algoritmos Gulosa e Pesquisa em Largura. O algoritmo Gulosa diferencia-se dos dois primeiro devido à sua opção de escolher a tarefa 6 em vez da tarefa 5. Isto acontece, porque o algoritmo escolhe a melhor solução local, sendo a tarefa 6 a melhor solução por necessitar de menos elementos.

A Pesquisa em Largura fornece resultados diferentes, escolhendo, de entre as tarefas em paralelo, a tarefa 4, por ser a primeira a ser visitada (a entrar na fila de nós visitados).

5 Conclusões

A análise do resultado das experiências recai sobre qual dos algoritmos devolve o melhor desempenho das tarefas. O algoritmo A*, em que o valor de cada sucessor é dado por $F(n)=G(n)+H(n)$, é uma mistura do algoritmo de Custo Uniforme ($F(n) = G(n)$) com métodos gananciosos ($F(n)=H(n)$), 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.

De facto, 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. Relativamente ao algoritmo Pesquisa em Largura, este algoritmo é óptimo caso todas as arestas possuam o mesmo custo/valor, no sentido que a primeira solução encontrada é necessariamente a melhor, e se for finito, é também um algoritmo completo.

Assim, podemos concluir que o A* encontra sempre a melhor solução para o problema, ao contrário do que acontece com os algoritmos Custo Uniforme, Ganancioso e Pesquisa em Largura.

O gráfico abaixo demonstra a duração da execução de cada um destes algoritmos para a mesma situação (muitas tarefas e poucos recursos), tendo sido verificado, pelas distintas funções de cálculo de cada sucessor, que o algoritmo A* é aquele que demora mais tempo a ser executado (considera o custo e heurística) e o algoritmo de Custo Uniforme o mais rápido (considera apenas o custo).

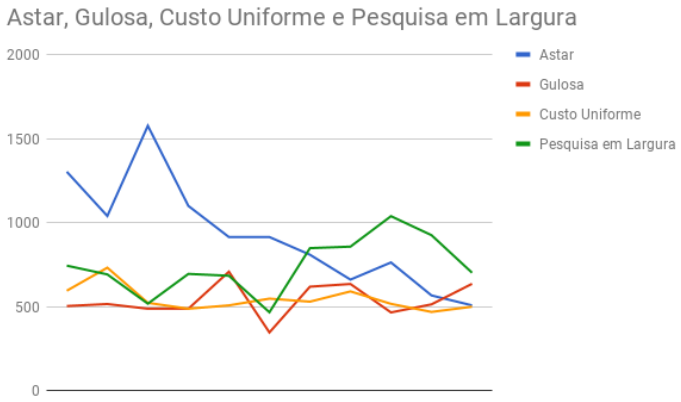

Figura 3: Gráfico de comparação do tempo de execução dos algoritmos (tempo apresentado em milisegundos)

Em suma, a realização deste 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*, Gulosa, Custo Uniforme e Pesquisa em Largura, que requerem um estudo complexo, mesmo antes de serem postos em prática. A acrescentar, que também foi possível consolidar os nossos conhecimentos e capacidade de análise em relação a problemas de pesquisa de soluções e estratégias de implementação dos algoritmos no contexto de alocação de tarefas a recursos.

6 Recursos

6.1 Bibliografia

[**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/ amit-p/GameProgramming/Heuristics.html.
[**3**] Greedy algorithm. Acedido em 2018. Disponível em https://en.wikipedia.org/wiki/Greedy_algorithm.

6.2 Percentagem de Trabalho dos elementos do grupo

Os elementos do grupo cooperaram entre si de forma a distribuir o trabalho uniformemente e para que houvesse partilha de sugestões e conhecimento. De uma forma organizada, foi planeada a divisão de tarefas de modo a responder eficientemente aos objetivos iniciais, colaborando, assim em todas as fases de implementação do projeto.

Diogo Reis - 33,(3)%
Francisca Cerquinho - 33,(3)%
Mariana Silva - 33,(3)%

7 Manual do utilizador

Para introduzir os dados do problema, o utilizador terá de escrever nos ficheiros *elements.txt* e *tasks.txt*.

No caso do ficheiro *elements.txt*, tem de introduzir todos os elementos da equipa separados por linha. Em cada linha o utilizador insere o nome do elemento, as suas competências e o seu desempenho em cada competência respetivamente. Por exemplo: "Pedro;[java,web];[4,3]", o que significa que o Pedro sabe java com um desempenho de 4 e web com desempenho de 3.

Para introduzir as tarefas a realizar o utilizador escreve no ficheiro *tasks.txt* todas as tarefas a serem realizadas separadas por linha. Em cada linha insere o nome da tarefa, número de elementos, competência necessária e os precedentes da tarefa.Por exemplo: "T4;2;c;[T3,T1]", o que significa que a tarefa T4 necessita de 2 elementos que saibam programar em c, tendo como precedências a tarefa T3 e T1.

Após a inserção dos dados o utilizador terá de correr o *script.sh*. Alternativamente o utilizador deve abrir a consola de comandos e navegar até a pasta do projeto, compilar o projeto usando: javac -d out/ -Xlint src/*.java, deslocar-se até à pasta out usando: -cd out/ e executar o projeto usando: java Interface tasks.txt elements.txt.

Após a execução do projeto, é escrito na consola a informação de todos os nós e arestas desse mesmo grafo sendo que cada nó contém uma tarefa, os seus respetivos requisitos e o elemento que foi escolhido para a realizar. Cada nó repetido representa as diferentes hipóteses de alocação de um elemento a uma mesma tarefa. Em relação às arestas, é mostrado ao utilizador o nó a que uma aresta se está a ligar e a informação desse mesmo nó destino juntamente com o nome e desempenho do elemento que realizou a tarefa anterior.

O utilizador até decidir sair do programa, pode escolher que algoritmo usar para resolução do problema, sendo apresentado um menu intuitivo com 5 opções diferentes, em que, cada opção é escolhida simplesmente escrevendo o número da opção desejada e pressionando *ENTER* no teclado. Se o problema em questão não tiver solução, o programa detetará o erro e enviará uma mensagem ao utilizador com o tipo de erro ou falta de dados para resolução do mesmo. Caso contrário, o programa mostra a solução ideal para o algoritmo escolhido.