Universidade do Porto

Faculdade de Engenharia

FEUP

Pesquisa na Alocação de Trabalhadores

Relatório Intercalar

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo, A2_3:

Hugo Drumond – 201102900 – hugo.drumond@fe.up.pt

João Loureiro – 200806067 – ei08101@fe.up.pt

Joel Carneiro – 201100775 – ei11053@fe.up.pt

16 de abril de 2016

Especificação

Descrição do trabalho

Este trabalho tem como objetivo determinar a melhor alocação de trabalhadores a diferentes tarefas.

É pretendida a contratação de trabalhadores para um conjunto de tarefas de programação. As tarefas estão agrupadas em módulos de software, sendo que as tarefas de um mesmo módulo necessitam de ser programadas recorrendo à mesma linguagem de programação. Por outro lado, cada tarefa faz uso de uma determinada tecnologia (por exemplo: *web-services*, *sockets*, algoritmos de IA, bases de dados, entre outros)

Cada trabalhador possui uma ou mais especializações em tecnologias e/ou é proficiente numa ou mais linguagens de programação. Alocar um trabalhador tem custo associado, este é proporcional às suas competências tecnológicas e à sua experiência.

Pretende-se obter a alocação de trabalhadores que, permitindo que todas as tarefas sejam executadas, permita minimizar o custo de toda a operação.

Devem ser utilizados métodos de pesquisa informada (nomeadamente A*) para efetuar a alocação pretendida. O programa deve permitir configurar as tarefas necessárias e os trabalhadores disponíveis, e deve ser testado com diferentes instâncias do problema.

Arquitetura do trabalho

Representação de conhecimento

A informação sobre os módulos e os trabalhadores, e todas as suas características, será guardada num ficheiro de formato *JSON*. Escolhemos este suporte, entre outros possíveis como é o exemplo das bases de dados ou *XML*, para representar a informação relativa a este projeto, pois este é leve, portável, legível, simples, com bastante suporte a nível de bibliotecas de *parsing* e oferece uma excelente representação para objetos de programação como *Arrays* e *Strings*.

```json
{
  "modules": [
    {
      "name": "module1",
      "tasks" : [
        {
          "name": "task1",
          "technology": "sockets"
        },
        {
          "name": "task2",
          "technology": "web-services"
        }
      ]
    }
  ],
  "workers": [
    {
      "name": "Hugo",
      "languages": [ "java", "cpp", "c" ],
      "technologies": [ "sockets", "web-services"],
      "years_of_experience": 3
    },
    {
      "name": "João",
      "languages": [ "java", "cpp", "c", "ruby", "scala", "python", "lua", "prolog" ],
      "technologies": [ "sockets", "web-services", "databases", "algorithms"],
      "years_of_experience": 3
    },
    {
      "name": "Joel",
      "languages": [ "java", "cpp", "c" ],
      "technologies": [ "sockets", "web-services", "algorithms"],
      "years_of_experience": 3
    }
  ]
}
```

Fig. 1 - Estrutura do ficheiro de input JSON

Arquitetura do trabalho

Em todos os problemas de procura de soluções é necessário especificar qual as características de um estado, isto é, a informação necessária à análise do problema em cada etapa. O estado neste problema será descrito da seguinte maneira:

- Lista de pares (tarefa, trabalhador);
- Lista das tarefas ainda não consideradas organizadas por módulo;
- Lista dos trabalhadores por alocar;
- Lista da linguagem atribuída a cada módulo.

As ações correspondem à atribuição de uma tarefa a um trabalhador. O primeiro par atribuído a um módulo define a linguagem deste. As ações possíveis dependem:

- das tarefas ainda disponíveis;
    - Intercepção da linguagem do módulo da tarefa T às linguagens dos trabalhadores disponíveis;
        - Intercepção do conjunto resultante com a tecnologia da tarefa T. Resulta nos pares {(T, Worker1), (T,Worker2), ... }

O custo dessa transição (ação) será definido em função do número de linguagens e tecnologias que um programador conhece e os anos de experiência no mercado de trabalho, por exemplo: transição = #n.linguagens + #n.tecnologias * 0.75 + n.experiencia * 0.25.

A função sucessor criará um estado novo para cada ação possível, e colocará cada um na lista de prioridades *fringe*, ordenado por ordem crescente de F(n). A solução é encontrada quando a lista de tarefas é nula. Que equivale a dizer que cada tarefa foi atribuída a um trabalhador.

A heurística tem de ser consistente para garantir a solução ótima. Pode-se dizer que na melhor das hipóteses o custo do nó atual até o objetivo, é o custo dos N trabalhadores disponíveis mais baratos. Sendo N o número de tarefas em falta.

Divisão do trabalho

Como anteriormente referido vamos guardar todos os ficheiros de dados em formato JSON. Consequentemente, numa das fases do trabalho proceder-se-á ao parsing e armazenamento da informação resultante numa estrutura de dados adequada ao tratamento pelo algoritmo A*.

Será dado mais foco às seguintes tarefas:

- Implementação genérica do algoritmo, com prevenção de exploração de nós já visitados através de um *Set* (Modo de pesquisa por grafo).
- Definição de uma heurística cujo custo heurístico num dado nó seja menor ou igual ao custo real para atingir um dado objetivo (Admissibilidade). Se pesquisa for feita em modo de árvore.
- Garantir que a heurística escolhida é consistente, isto é, a admissibilidade em conjunto com a não exploração de nós já visitados poderá não levar à solução ótima. Em suma, é necessário garantir que h(anterior) <= c(anterior->novo) + h(novo) para pesquisa em modo de grafo. Consistente -> Admissível.
- Criação de um conjunto de ficheiros de teste.
- Comparação entre diferentes heurísticas.

Trabalho efetuado

Descrição do trabalho

Nesta etapa do trabalho, preferiu-se dedicar mais tempo questionando e debatendo como o trabalho seria organizado e implementado. Foram definidas as tarefas que o grupo irá desenvolver tal como os módulos de software, ferramentas, estratégias para a verificação de resultados e ideias para a definição de uma heurística. A pesquisa irá ser feita em modo de grafo. Por essa razão a heurística terá de ser consistente.

Resultados esperados e forma de avaliação

Testes a definir para validar o trabalho

A validação será feita através da interpretação dos resultados de *output* gerados a partir de uma população presente num ficheiro de teste. Para ficheiros relativamente pequenos fazer-se-à um contraste com uma possível solução por nós encontrada. O resultado será imprimido para a consola de uma maneira estruturada, legível e de fácil compreensão.

Conclusões

Com este trabalho será possível a aprendizagem de um dos métodos de pesquisa informada mais conhecidos e polivalentes, o algoritmo A*. Resultante da combinação do algoritmo *greedy*(h) e *uniform-cost-search*(g), ou seja, estimativas de custos posteriores e custos acumulados. $F(n) = h(n) + g(n)$.

Chegou-se à conclusão que a dificuldade nestes tipos de algoritmos centra-se na descoberta de uma heurística que garanta o resultado ótimo. E que, algoritmos semelhantes só diferem na ordem pela qual estados por explorar são colocados na lista de prioridades *(fringe)*.

Recursos

Software

Optou-se por codificar o algoritmo em java visto ser uma linguagem estável, bem documentada, amplamente usada, e com bibliotecas de grande qualidade. Uma outra razão, é

o facto de existir imenso software de desenvolvimento com bastantes *features*. Daí, escolhermos o *Intellij IDEA*.

Será também utilizada uma base de dados *JSON* para gerenciar os dados referentes aos trabalhadores e tarefas. Das muitas bibliotecas encontradas foi escolhida a *biblioteca JSON-java*.

O código fonte e os ficheiros auxiliares serão colocados num repositório *Git* de modo a facilitar a interação entre os elementos do grupo.

Bibliografia

Documentação de software

- http://stleary.github.io/JSON-java/index.html

Vídeos

- https://www.youtube.com/playlist?list=PL-XXv-cvA_iA4YSaTMfF_K_wvrKAY2H8u