Universidade do Porto

Faculdade de Engenharia

FEUP

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Otimização da gestão de projetos

Authors:

Duarte Pinto
- up201304777 - up201304777@fe.up.pt

Filipa Ramos
- up201305378 - up201305378@fe.up.pt

Gustavo Silva
- up201304143 - up201304143@fe.up.pt

9 de Fevereiro de 2017

Conteúdo

1 Introdução

No âmbito da unidade curricular de Inteligência Artificial pretende-se desenvolver um programa que, com base em algoritmos genéticos e arrefecimento simulado, faça a gestão de um projeto balançando os elementos participantes e as tarefas a realizar do mesmo. O sistema é composto por um conjunto de tarefas que pertencem ao projeto em análise e um conjunto de elementos que as podem realizar. A gestão a ser realizada tem em vista minimizar o tempo ocupado para satisfazer todas as tarefas do projeto usando a melhor combinação de elementos para cada tarefa. Será feita uma análise comparativa entre o desempenho das solução encontradas com algoritmos genéticos e arrefecimento simulado.

Os objetivos principais do projeto passam pela exploração da implementação prática dos algoritmos genéticos e do algoritmo de arrefecimento simulado. O processo de análise comparativa irá fomentar o conhecimento adquirido, evidenciando as vantagens principais de cada algoritmo e as suas dicotomias fundamentais.

Espera-se que surjam dificuldades na implementação prática dos algoritmos estudados teoricamente, principalmente na construção dos cromossomas pois existem dúvidas em relação à sua influência na eficiência da solução encontrada. Para além disto, a melhor adaptação da função de avaliação ao problema por forma a obter os melhores resultados revela-se um processo tumultuoso. Os membros decidiram optar por otimizar o tempo utilizado a concluir todas as tarefas do projeto em estudo. Desta forma, a melhor solução será a que implicará um menor tempo de conclusão do projeto em questão.

2 Especificação

2.1 Problematização

O sistema tem por objetivo otimizar a atribuição de membros por tarefas num dado projeto. Os dados do mesmo são introduzidos por input através de um ficheiro.

Um projeto em análise caracteriza-se por um conjunto de tarefas (Task) a cumprir, tendo estas um nome (por motivos de identificação) e uma duração. Existe ainda um conjunto de elementos (Element) que podem ser atribuídos a essas mesmas tarefas. Um elemento é identificado por um nome e tem uma lista de competências (Skill) avaliadas em função da sua capacidade. Por exemplo, o elemento "João"tem competências na área da informática a um nível 0.6 e na área da economia com nível 1.0 (máximo).

2.2 Aquitetura

O projeto foi dividido em três "packages"principais que representam os três níveis mais importantes. Um dos packages diz respeito às classes que guardam informação sobre as tarefas, os elementos e as competências. Os outros dois dizem respeito à implementação do algoritmo genético ou do arrefecimento simulado.

A arquitetura pensada para o projeto passa pela implementação de três classes principais que interagem entre si próprias. A classe "Task"representa uma tarefa e guarda a sua duração. Um elemento é representado pela classe "Element"que mantém a identificação do mesmo. A classe "Skill"corresponde a uma competência. Estas ligam-se entre si por forma a que um elemento tenha vários skills e um skill tenha várias tarefas tal como é visível na figura 1. A relação que se pretende obter será a que liga os elementos a tarefas.

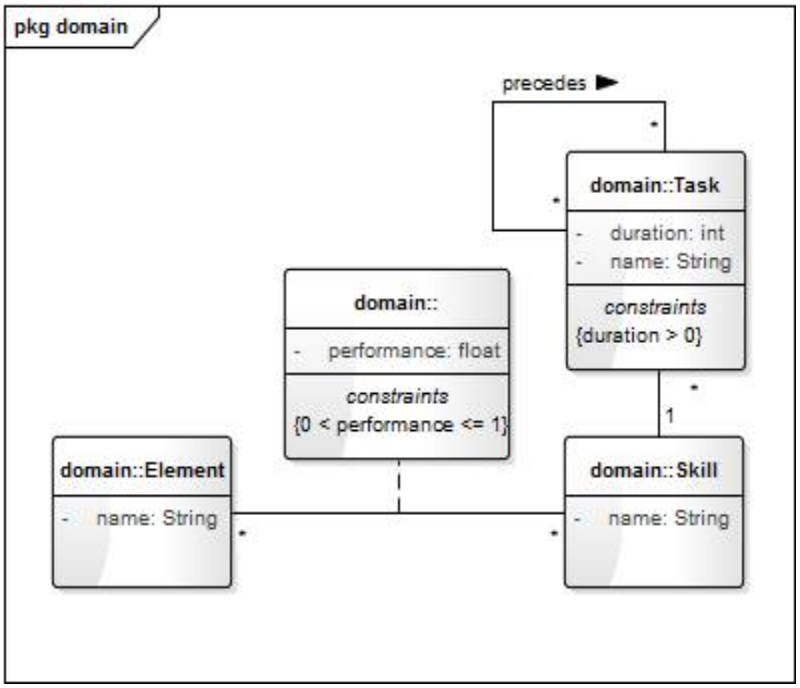

Figura 1: Diagrama de classes UML

2.3 Formato do input

O input será colhido de um ficheiro de formato *json* e será estrutrurado conforme está representado no exemplo apresentado a seguir.

```
{
  "skills":
     [ "Z", "Y" ],
   "tasks": [
     { "name": "A", "duration": 5,"skill": 0,"precedences": []},
     { "name": "B", "duration": 3,"skill": 1,"precedences": [0]}],
   "elements": [
      { "name": "Duarte Pinto", "skills": [[0,0.5], [2,0.1] ]},
      { "name": "Filipa Ramos", "skills": [[0,1.0]]},
      { "name": "Gustavo Silva", "skills": [[1,0.5], [2,0.1] ]}]
}
```

2.4 Fases

O projeto será dividido em fases de trabalho. A inicial passa pelo desenvolvimento da arquitetura supracitada. Seguidamente, proceder-se-á implementação dos algoritmos genéticos. Finalmente, será desenvolvido o arrefecimento simulado. Após terem sido terminados ambos os algoritmos será realizada a análise comparativa entre os dois, sendo esta a fase final do projeto.

2.5 Algoritmos Genéticos

2.5.1 Estrutura do Cromossoma

A construção dos cromossomas segue a ordem das precedências de tarefas. Assim, o mesmo é composto pelo id da tarefa seguido de 0 ou 1 para cada elemento, sendo que 1 significa que o membro trabalhou na tarefa e 0 significa que o elemento não trabalhou na tarefa. Desta forma, o cromossoma final será composto por x blocos, sendo que x é o número de tarefas existentes. Cada bloco terá tamanho variável dependente do projeto em análise - conforme o número de elementos total e o número de bits necessários para representar a tarefa com um identificador. A fórmula utilizada para cálculo do número de bits máximo do identificador é a apresentada na equação 1.

$$\lfloor(\frac{\log_{10}(size-1)}{log_{10}(2)+1}\rfloor \tag{1}$$

Figura 2: Fórmula para calcular o número de bits máximo para representar com um id binário todas as tarefas (size equivale ao número de tarefas).

Por exemplo, se o projeto em análise tiver duas tarefas e dois membros um dos cromossomas da população poderá ser o representado na figura 3. Neste caso teríamos os membros 1 e 3 alocados na primeira tarefa e, na segunda, apenas o membro 2. A tarefa com id 0 teria precedência à tarefa com id 1 ou seriam concorrentes.

Figura 3: Cromossoma exemplificativo

2.5.2 Função de Avaliação

A função de avaliação implementada procura avaliar os cromossomas da população tendo por base o tempo total de compleção das tarefas todas do projeto. Assim, um cromossoma com menor valor de *fitness* será mais apto visto que terminou as tarefas em menor tempo. Esta função trata variadas situações extraordinárias. No caso de haverem duas tarefas sem precedências entre si, é procurada a primeira tarefa que possa ser executada de acordo com a ordem de prioridades definida no cromossoma. Para além disto, se houver tarefas repetidas, apenas é considerada a sua primeira aparição no cromossoma. Se houver tarefas que não apareçam no cromossoma considera-se que elas estão no fim da lista de prioridades do cromossoma.

2.5.3 Seleção

A seleção é feita por uma roleta aleatória que gera valores como está representado no excerto de código a seguir. Estes valores implicam um valor aleatório multiplicado pelo *fitness* total da população. A este valor é subtraído o do *fitness* individual de cada cromossoma. Os selecionados são os que têm menor valor final. A seleção elitista pode implicar um cromossoma ou mais, sendo este valor escolhido conforme o que se pretende testar.

```
for (int i = 0; i < amountToSelect; i++) {
   double d = random.nextDouble() * population.getTotalFitness();
   for(int j = 0; j < population.getSize(); j++) {
      d -= population.getChromosome(j).getFitness();
      if (d <= 0){
         selected[i] = population.getChromosome(j);
         break;}}
}
```

2.5.4 Cruzamento

O cruzamento será aplicado de três maneiras distintas, sendo feita uma análise da eficiência de cada um dos métodos por forma a obter o mais eficaz. Assim, espera-se testar os seguintes métodos:

1. Cruzamento entre tarefas

   Troca de tarefas entre cromossomas incluindo o id e os elementos.

2. Cruzamento dos elementos entre tarefas

   Troca dos elementos de uma tarefa com os elementos de outra tarefa noutro cromossoma.

3. Cruzamento de elementos da mesma tarefa

   Troca de elementos de uma tarefa com os elementos da mesma tarefa de outro cromossoma.

2.5.5 Mutações

A mutação ocorre a uma taxa variável, sendo esta selecionada conforme os testes pretendidos. A mutação exclui os cromossomas elitistas, tendo por base o princípio de que um cromossoma elitista tem a melhor seleção de genes e, após uma mutação, essa combinação pode ser alterada para uma pior. São gerados valores aleatórios para cada cromossoma numa roleta e, se este for menor que o valor da taxa de mutação, é trocado o bit que resulta do resto da divisão da posição do mesmo no cromossoma pelo comprimento total do cromossoma.

2.6 Arrefecimento Simulado

Quando adaptado de maneira eficiente o algorítmo de Arrefecimento Simulado é característico pela facilidade de implementação e pela rapidez de convergência do resultado.

Para este projecto optamos por representar a nossa solução através de uma lista de tasks.

2.6.1 Representação

A representação da solução é importante pois tem que permitir a geração rápida do próximo estado e rápido calculo do ΔE.

Para tal colocamos as tarefas numa lista ordenada de tasks onde cada task aparece numa posição depois de todos os seus antecessores e antes dos seus sucessores e onde as tasks mais pequenas têm prioridade(aparecem primeiro na lista) sobre as tasks mais pesadas.

2.6.2 Atribuição dos tempos iniciais

Para atribuir os tempos iniciais de cada tarefa executamos um ciclo onde cada iteração representa a passagem de um *sprint*, de duração é varíavel. Começamos com $i = 0$, percorremos a lista e tentamos atribuir o tempo inicial à task j. Se as tasks antecessoras de j já tiverem terminado e houverem pessoas com as *skills* para executar a tarefa, então o $j_{start\_time} = i$. Caso contrário, incrementamos i e voltamos a verificar se é possível fazer a atribuição de tempo inicial à task. Nenhuma task pode ter o seu tempo inicial atribuído que a task anterior na lista ordenada também tenha o tempo inicial atribuído. Ao atribuir o tempo inicial a uma task é automaticamente calculado quando é que esta irá terminar, com base no número de pessoas com as skills necessárias para completar a task disponíveis. Essas pessoas ficam então alocadas à task e não podem ser utilizadas noutras tasks enquanto a task não tiver terminado.

```
1  int i = 0;
2  for(Task j : tasks){
3     while(true){
4        if(j.assigned)
5           break;
6        if(j.predecessorsFinished() && workers.hasAvailableResources(j)){
7           workers.assign(j,i);  //Assigns resources(people) to the task j from i until
               the end of the task
8           j.assignStartTime(i);}
9        else {i++;}
10    }
11 }
```

2.6.3 Geração do próximo Estado

O próximo estado é gerado através de uma lista ordenada víavel da seguinta maneira:

1. É escolhido uma task aleatória e são calculadas as posições da task antecessora mais recente e da task sucessora mais próxima.
2. É atribuída à task uma posição aleatória no intervalo das duas posições anteriormente calculadas.
3. Se a atribuição da nova posição for válida, é efectuado um deslocamento de todas as tasks que se encontram entre a antiga e a nova posição.

Depois de efectuados estes passos obtém-se a uma lista que representará o próximo estado.

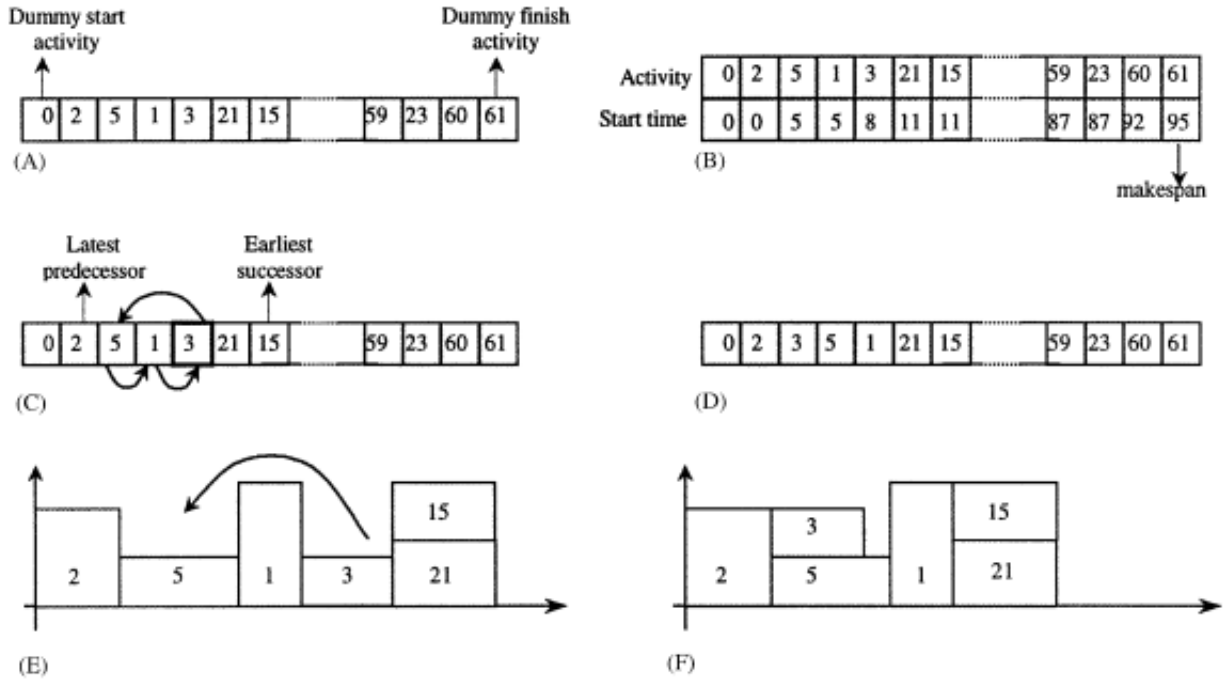

Figura 4: Representação de uma lista e processo de geração de um novo estado. (A) Representação de uma lista ordenada de tasks. (B) Exemplo de um planeamento viável com base na lista de tasks. (C) Geração aleatório de uma nova posição para a task 3. (D) Nova lista após o deslocamento. (E) Representação gráfica do Estado Inicial. (F) Representação gráfica do Novo Estado.

3 Trabalho Realizado

As fases abordadas até ao presente relatório foram as duas primeiras. A arquitetura explicitada pelo diagrama de classes uml já foi implementada e a obtenção de solução por algoritmos genéticos encontra-se numa fase avançada. De facto, o grupo encontra-se a afinar o algoritmo genético implementado através de testes por forma a maximizar a sua eficiência.

4 Testes

Por forma a garantir a integridade do sistema e tendo em vista a obtenção da melhor solução possível pretende-se aplicar testes intensivos e promenorizados forçando situações complexas para visionar a resposta do programa. Espera-se criar projetos com 30, 60 e 120 tarefas e, sobre estes, será feita a análise comparativa para observar o comportamento do sistema em situações de diferente complexidade. Outras opções analisadas serão projetos com mais tarefas do que membros e vice-versa. Isto permite uma melhor apreensão da extensão da solução criada. Para além disto, o sistema será observado em situações de projetos com membros com listas de competências iguais ou diferentes. Finalmente, serão esmiuçadas situações com projetos apenas com tarefas sem precedências ou simplesmente tarefas com precedências.

5 Conclusões

Em suma, verifica-se que o projeto se encontra a ser desenvolvido a um ritmo coeso e coerente, o que indica uma progressão saudável do mesmo. As fases encontram-se claramente delineadas o que permite uma maior eficácia na distribuição de tarefas entre os membros do grupo e, ao que tudo indica, levará à construção de um projeto bem documentado e estruturado.

O projeto escolhido despoletou a curiosidade de todos os membros. Apesar de por vezes serem encontradas dificuldades, são estas que dão alento ao estudo mais aprofundado das matérias abordadas e permitem um avanço sólido do mesmo. Espera-se que o trabalho a realizar no futuro se encontre dentro dos paradigmas do que já se encontra feito.

Referências

[1] Tao Zhang Carl K. Chang, Mark J. Christensen. Genetic algorithms for project management, 2001.

[2] H. Lecocq K. Bouleimen. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, 2002.