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

29 de Maio de 2016

Conteúdo

Introdução

«The scheduling of tasks and the allocation of resource in medium to large-scale development projects is an extremely hard problem and is one of the principal challenges of project management due to its sheer complexity.»[1]

No âmbito da unidade curricular de Inteligência Artificial foi desenvolvido um programa de otimização que tem por objetivo gerir projetos. Tendo por pressuposto a citação explicitada acima foi implementado um sistema para alocar membros a um projeto da maneira mais eficiente tendo em consideração a duração de cada tarefa e as competências de cada membro para cada tarefa. Pretende-se no presente documento avaliar comparativamente os resultados demonstrados por cada

[1] Carl K. Chang et al, *Genetic Algorithms for Project Management*, (Holanda: Kluwer Academic Publishers, 2001)

um dos algoritmos de otimização usados - algoritmos genéticos e arrefecimento simulado. Inicialmente será esmiuçado o problema sugerido e a perspetiva das soluções implementadas. Nesta especificação será englobada uma análise detalhada ao tema e aos cenários problemáticos que surgiram nessa mesma análise. A abordagem técnica adaptada de forma a solucionar estes obstáculos será pormenorizadamente descrita.

Um dos objetivos principais do presente documento é avaliar a resposta de ambos os algoritmos em cenários semelhantes por forma a melhor compreender a sua eficiência e as suas situações de risco. As conclusões retiradas das experiências efetuadas serão indispensáveis à exploração das possibilidades e fraquezas de cada algoritmo. Para além disto, surgem objetivos secundários tais como o aprofundamento do estudo de métodos de inteligência artificial e o estudo da aplicação prática da teoria abordada nas aulas.

Especificação

Problematização

«Um projeto é constituído por um conjunto de tarefas a desenvolver por um ou mais elementos. As tarefas podem ter precedências entre si e têm uma duração (pessoa/mês). Cada elemento candidato possui um conjunto de competências, que cobrem uma ou mais tarefas. É conhecido ainda o desempenho de um elemento em cada uma das suas competências.»[2]

Como é especificado em cima 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 no formato json no qual estão representadas as tarefas, elementos e as competências de cada um deles.

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.

Cenários

O programa desenvolvido teria incomensurável utilidade para qualquer tipo de empresas. Imagina-se cenários onde o software seria uma mais valia como o processo de iniciação de um projeto na qual não é claro como deve ser feita a divisão de tarefas entre os membros. Para uma empresa da área informática seria da maior importância visto que para construir software são precisos grupos de trabalho em que as competências de cada um são diferentes. É ainda aplicável no meio académico, não só como ferramenta mas também como estudo de algoritmos de otimização. Trazendo o problema para o âmbito real leva a que este possa ser aplicado em contextos mais extensivos enriquecendo o contributo dado pelo mesmo.

[2]Enunciado do problema.

Dificuldades

Um dos maiores obstáculos encontrados no planeamento da implementação foi o jogo de todas as variáveis do problema da forma mais eficiente. A combinação do uso da duração das tarefas com as suas precedências e das competências dos elementos provou-se como um desafio a ultrapassar.

A maior dificuldade identificada nos algoritmos genéticos foi a da construção de uma função de avaliação que tivesse em conta a precedência de tarefas. Assim, inicialmente, pensou-se ser necessária uma heurística de escolha de ordem de tarefas. Se este processo não fosse bem otimizado a solução apresentada não seria a melhor possível. Logo, esta heurística teria de ser cautelosamente planeada. Contudo, optou-se por implementar a estrutura do projeto de forma diferente sendo que a ordem das tarefas seria transmitida pelos cromossomas e seria decidida aquando da sua construção. A implementação será esmiuçada mais à frente. Para além disto, a escolha de um método de cruzamento provocou algumas dúvidas nos elementos porém após terem sido realizados alguns testes foi escolhido layout que produzia mais frequentemente melhores resultados.

No algorítmo de arrefecimento simulado a maior dificuldade foi encontrar uma representação que permitisse a geração de novos estados e alocação de elementos às tasks de maneira eficiente e rápida.

Datasets

Formato do input

O input é colhido de um ficheiro de formato *json* e tem a estrutura conforme representado no exemplo apresentado a seguir.

```
{
   "skills": [
      "JQuery", "PHP", "CSS" ,"HTML"
               ],
   "tasks": [
      {"name":"Construir pagina", "duration": 8,"skill": 3,"precedences":[]},
      {"name": "Estilizar pagina", "duration": 5,"skill": 2,"precedences": [0]},
      {"name": "Fazer login", "duration": 3,"skill": 1,"precedences": [0,1]},
      {"name": "Animacoes", "duration": 7,"skill": 0,"precedences": [0,1]}
            ],
   "elements": [
      {"name": "Duarte Pinto", "skills": [[0,0.5], [2,0.3] ]},
      {"name": "Filipa Ramos", "skills": [[0,1.0], [3,0.1] ]},
      {"name": "Gustavo Silva", "skills": [[1,1.0], [2,0.1] ]}
               ]
}
```

Algoritmos

Na presente secção são analisados os algortimos usados passando pela explicação da implementação decidida e pela descrição da perspetiva tomada ao enfrentar os mesmos. Inicialmente serão

esmiuçados os algoritmos genéticos, sendo explicada a função de avaliação implementada, a estrutura do cromossoma construído e os processos de seleção, cruzamento e mutação. Finalmente é explicitado o arrefecimento simulado incluindo a representação escolhida, a atribuição de tempos iniciais e a geração do próximo estado.

Atribuição do Tempo inicial e dos Elementos às Tarefas

Para atribuir os tempos iniciais e tempos finais 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 houver pelo menos um elemento 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.

```
int i = 0;
for(Task j : tasks){
   while(true){
      if(j.assigned)
         break;
      if(j.predecessorsFinished() && workers.hasAvailableResources(j)){
         workers.assign(j,i);  //Assigns resources(people) to the task j from i until
             the end of the task
         duration = getDuration(j, i, workers);
         end = i + duration
         j.assignStartTime(i,end);

      }
      else {i++;}
   }
}
```

Calculo de duração de uma tarefa

A duração de uma tarefa é dada em função do número de elementos e a *performance* de cada um dos elementos atribuídos à tarefa. A duração da tarefa é avaliada da seguinte maneira:

$$d_{total} = \frac{1}{\frac{1}{d_a} + \frac{1}{d_b} + \dots} \tag{1}$$

d_a , d_b ,... - tempo que os elementos a e b demoram a completar a tarefa

Sabendo que o tempo que cada elemento demora a concluir a tarefa que é dado pela função:

$$d_x = \frac{t}{p_x} \tag{2}$$

t - duração base da tarefa
t_x - performance do elemento x a efectuar a tarefa

Daqui podemos concluir que:

$$d_{total} = \frac{t}{p_a + p_b + \dots} \tag{3}$$

Função de Avaliação

Para analisar uma calendarização gerada e poder comparar com outras geradas é utilizado o valor de quando termina a última tarefa a ser completada, ou seja, o tempo total que demora a terminar todo o conjunto de projectos na dada calendarização

```
1  public float eval(Schedule schedule){
2     float bigger = 0;
3     for(Task task : schedule.getOrderedTasks()){
4        float completionTime = schedule.getTaskCompletionTime(task);
5        if(completionTime > bigger){
6           bigger = completionTime;
7        }
8     }
9     return bigger;
10 }
```

Algoritmos Genéticos

Estrutura do Cromossoma

Um cromossoma válido traduz a ordem de realização das tarefas sendo que a que aparece primeiro será realizada antes de todas as outras. Cada bloco de cromossoma contém um identificador que representa a tarefa e um bit para cada elemento do projeto. Os bits seguintes dizem respeito aos elementos. Se o elemento estiver alocado à tarefa o seu bit estará a 1. Por exemplo, se tivermos um projeto com 4 tarefas e 6 membros um dos cromossomas possiveis seria o seguinte representado na figura **??**. Pode-se observar que o primeiro elemento trabalhará em todas as tarefas menos a de id = 011. O elemento 5 só será alocado à primeira tarefa.

O cálculo do compimento do cromossoma depende da quantidade de membros e de tarefas do projeto. A fórmula aplicada por forma a obter o número de bits necessários à representação do identificador traduz-se na equação 1. No caso do cromossoma em cima o número de bits é 3. Assim, o comprimento de um bloco seria de 9 bits. Sendo que este tem 4 tarefas o comprimento total do cromossoma seria de 36 bits.

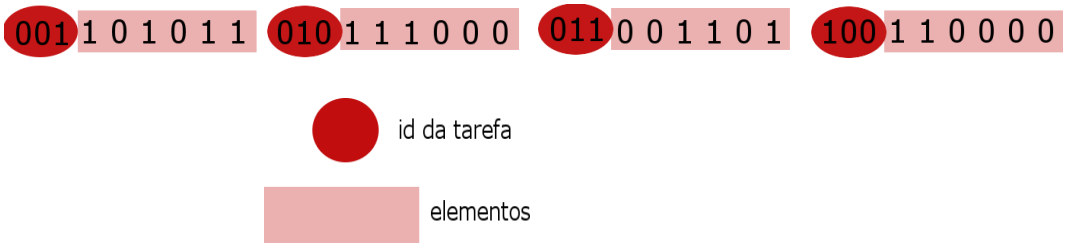

Figura 1: A figure with two subfigures

$$\lfloor (\frac{\log_{10}(size-1)}{log_{10}(2)+1} \rfloor \tag{4}$$

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).

Função de Avaliação

A função de avaliação verifica se o cromossoma é válido em termos de precedência de tarefas. Para além disto, calcula os tempos de início e de fim para cada membro alocado à tarefa. Verifica também se outra tarefa pode ser feita ao mesmo tempo. Se nenhuma tarefa puder ser feita, aumenta o tempo até à primeira a poder ser executada. Caso as condições não sejam cumpridas é aplicada uma penalização que se verifica pelo aumento do tempo. Assim, quanto menor o valor de **fitness** melhor o cromossoma.

Seleção

A seleção é feita por uma roleta aleatória que gera valores. 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.

Cruzamento

Foram escolhidos dois métodos de cruzamento que se revelaram mais eficientes. Um deles consiste na troca de membros de uma tarefa dentro de um cromossoma. É trocado apenas um bit dentro de tarefas aleatoriamente. O outro troca elementos de uma tarefa entre dois cromossomas distintos.

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.

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.

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.

Geração do estado inicial

O Primeiro estado é gerado percorrendo a lista de tasks e tentando adicionar a uma lista ordenada. Se os precedentes ainda não tiverem sido colocados na lista, chama a função recursivamente, passando o precedente como argumento, e só depois de todos os precedentes estarem na lista é que coloca a tarefa.

```
1  public List<Task> orderTask(List<Task> unorderedTasks){
2     List<Task> orderedTasks = new ArrayList<Task>();
3     for(Task task : unorderedTask){
4        if(!orderedTasks.contains(task)){
5           insertTask(orderedList, task);
6        }
7     }
8     return orderedTasks;
9  }
10
11 public void insertTask(List<Task> orderedList, Task task){
12    for(Task precedence : task.getPrecedences()){
13       if(!orderedList.contains(precedence)){
14          insertTask(orderedList,precedence);
15       }
16    }
17    orderedList.add(task);
18 }
```

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.

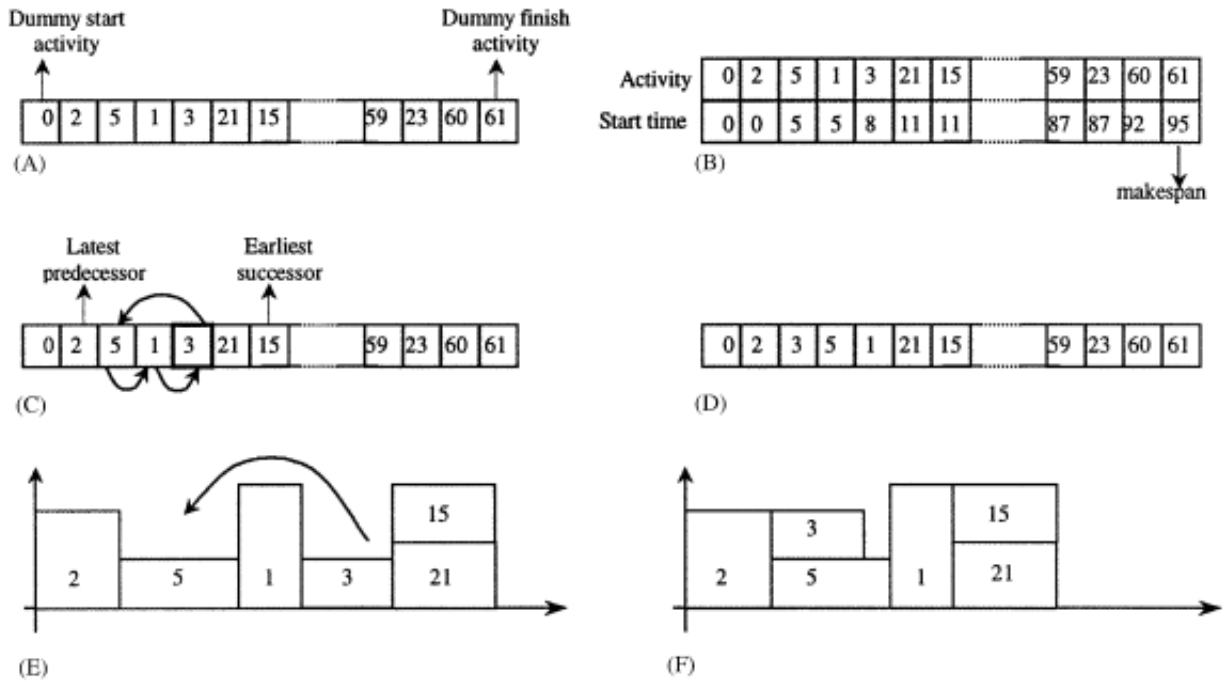

Figura 3: Representação de uma lista e processo de geração de um novo estado. (A) Representação de uma lista ordenada de tasks. (B) Exemplo de um planeamento viável com base na lista de tasks. (C) Geração aleatório de uma nova posição para a task 3. (D) Nova lista após o deslocamento. (E) Representação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

Temperatura

A temperatura vai descendo sempre gradualmente, tendendo para 0 ao longo do tempo

$$T_n = \alpha \times T_{n-1} \tag{5}$$

α - cooling rate

Desenvolvimento

Ferramentas, linguagens e ambientes

Para tornar mais fácil a visualização dos resultados foi usada uma API de visualização de grafos chamada NOME DA API. A linguagem usada foi Java.

Aquitetura

As classes indispensáveis à resolução do problema através de algortimos genéticos são as presentes no package `optimizer.solver.genetic_algorithm` e as suas relações e métodos principais estão descritos na figura 4.

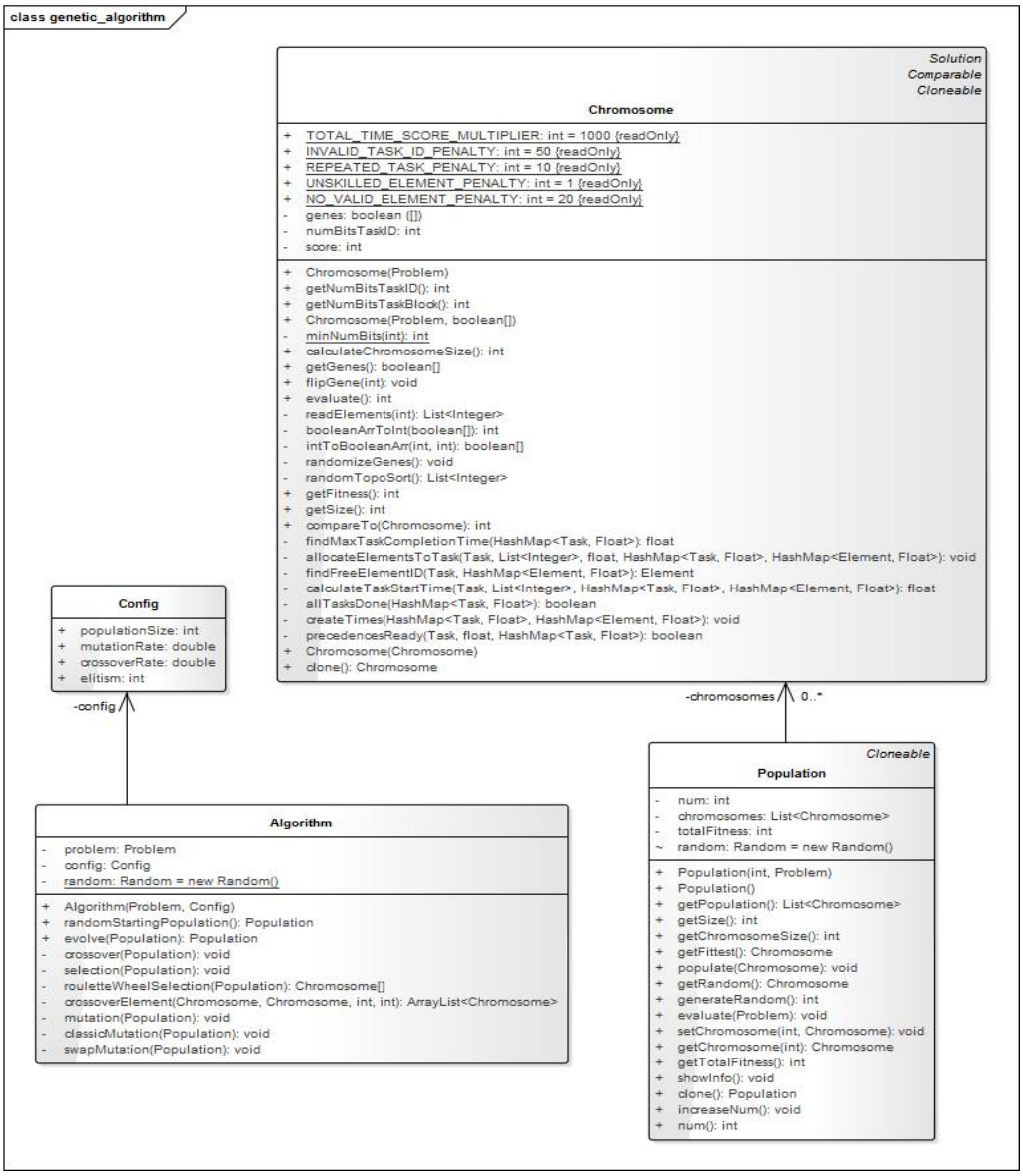

Figura 4: Diagrama de classes UML do package `optimizer.solver.genetic_algorithm`.

O projeto foi dividido em packages de forma hierárquica para representar os níveis da solução implementada. Um dos packages serve o propósito de guardar a informação relativa aos membros, tarefas e competências. Outro engloba dois outros packages onde está implementada a solução usando cada um dos algoritmos. O package .gui contém todas as classes que permitem a visualização gráfica da solução encontrada. Esta arquitetura de subpackages foi escolhida por forma a melhor representar a dimensão do problema e da implementação. Na realidade, a hierarquia é o que melhor representa a ligação entre os packages e as suas classes. Esta relação pode ser visualizada na figura 5.

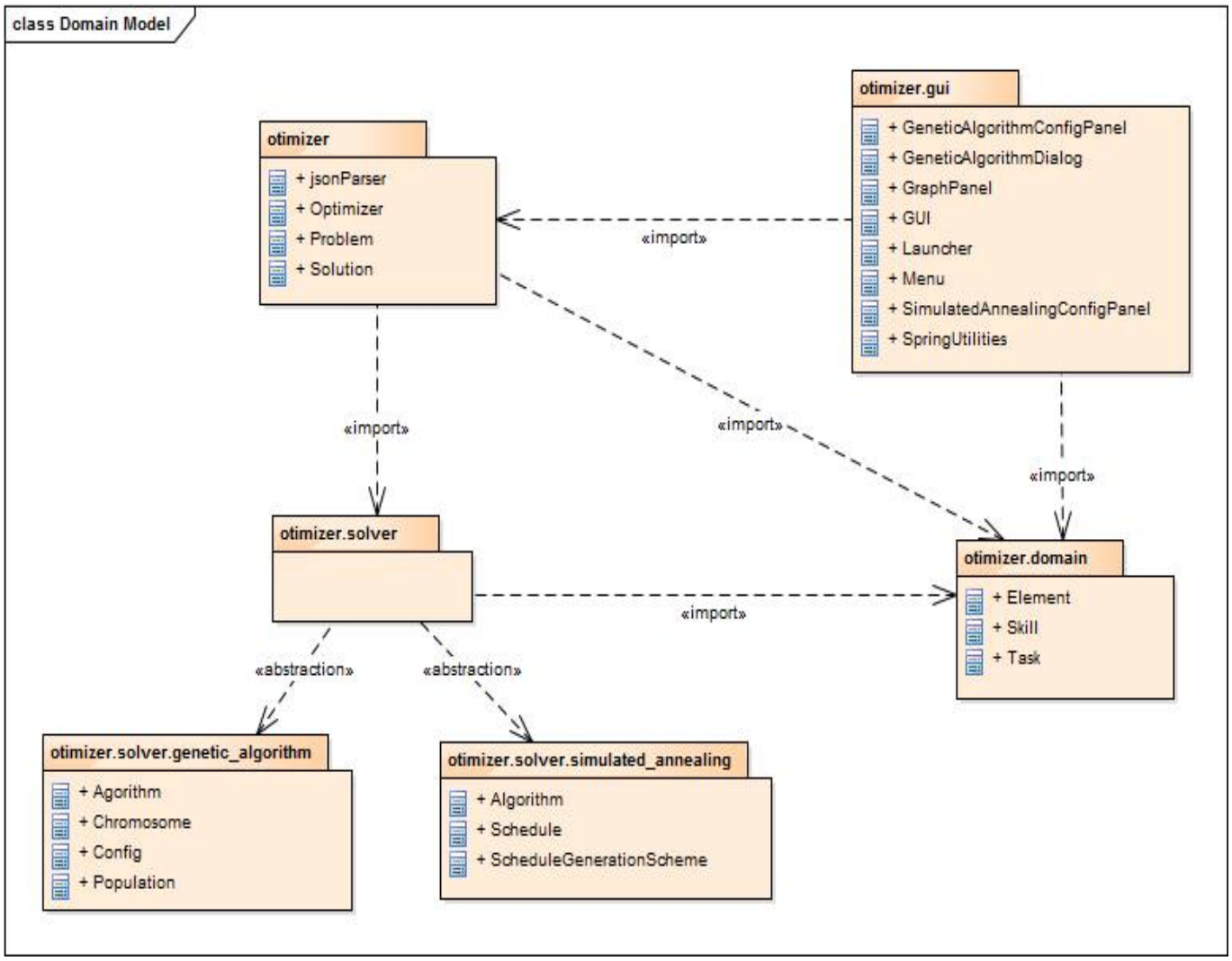

Figura 5: Diagrama de packages UML

Experiências

Experiência 1

O ficheiro usado para a experiência 1 foi o seguinte.

```
{
"skills": [
     "skill-A", "skill-B", "skill-C" ,"skill-D", "skill-E"
               ],
  "tasks": [
     {"name": "A", "duration": 8,"skill": 3,"precedences":[]},
     {"name": "B", "duration": 5,"skill": 2,"precedences": [0]},
     {"name": "C", "duration": 3,"skill": 1,"precedences": [0,1]},
     {"name": "D", "duration": 7,"skill": 0,"precedences": [0,1]},
     {"name": "E", "duration": 1,"skill": 2,"precedences":[]},
```

```
    {"name": "F", "duration": 2,"skill": 0,"precedences":[]},
    {"name": "G", "duration": 6,"skill": 2,"precedences":[1, 3]},
    {"name": "H", "duration": 3,"skill": 4,"precedences":[9]},
    {"name": "I", "duration": 4,"skill": 1,"precedences":[]},
    {"name": "J", "duration": 5,"skill": 3,"precedences":[4, 5]},
    {"name": "K", "duration": 5,"skill": 2,"precedences":[]},
    {"name": "L", "duration": 5,"skill": 2,"precedences":[7]},
    {"name": "M", "duration": 5,"skill": 1,"precedences":[9]},
    {"name": "N", "duration": 5,"skill": 0,"precedences":[6, 7]},
    {"name": "O", "duration": 5,"skill": 2,"precedences":[]},
    {"name": "P", "duration": 5,"skill": 3,"precedences":[1, 10]},
    {"name": "Q", "duration": 5,"skill": 1,"precedences":[13, 14]},
    {"name": "R", "duration": 5,"skill": 0,"precedences":[15]},
    {"name": "S", "duration": 5,"skill": 2,"precedences":[]},
    {"name": "T", "duration": 5,"skill": 3,"precedences":[3, 18]},
        ],
  "elements": [
    {"name": "Duarte", "skills": [[0,0.5], [2,0.3] ]},
    {"name": "Filipa", "skills": [[0,1.0], [4,0.1] ]},
    {"name": "Gustavo", "skills": [[3,1.0], [4,0.1] ]},
    {"name": "Joao", "skills": [[2,1.0], [3,0.1] ]},
    {"name": "Steve", "skills": [[3,1.0] ]},
    {"name": "Helder", "skills": [[4,1.0] ]},
    {"name": "Gugu", "skills": [[2,1.0], [3,0.1] ]},
    {"name": "Sansa", "skills": [[0,1.0], [1,0.1] ]},
    {"name": "Arya", "skills": [[4,1.0] ]},
    {"name": "Daenerys", "skills": [[1,1.0], [2,0.1], [3,0.2] ]}
          ]
}
```

Experiência 1 - A

Algoritmo Genético

As condições em que foi testado o ficheiro teste1.json foram as visíveis na figura 6.

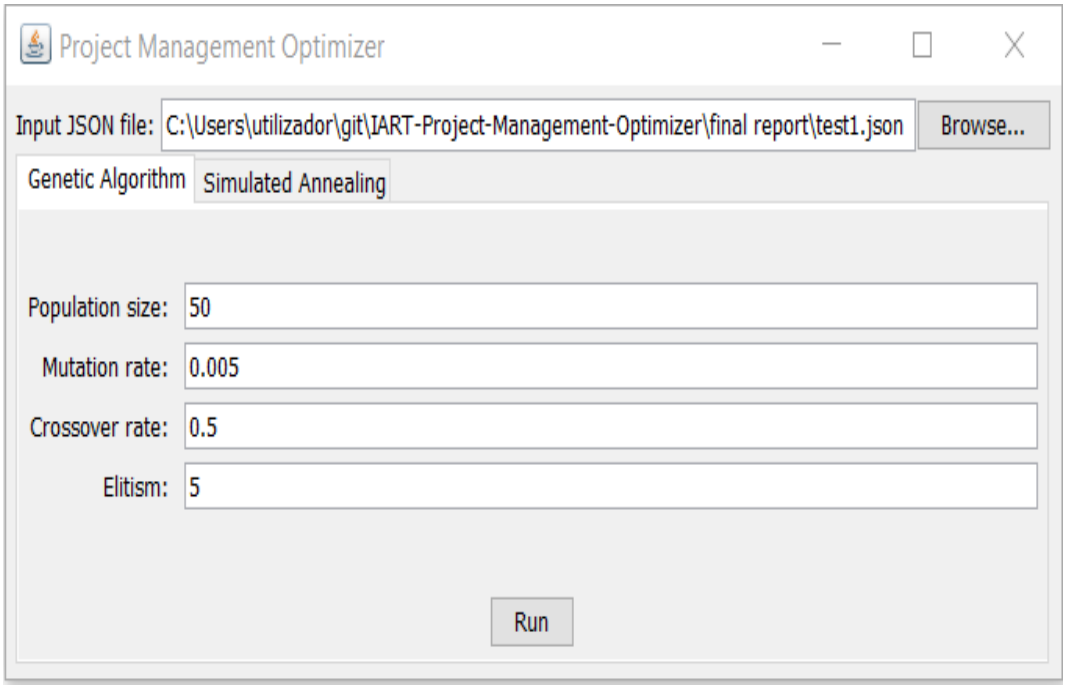

Figura 6: Condições do ficheiro teste1.json.

Os resultados obtidos foram os apresentados na figura 7.

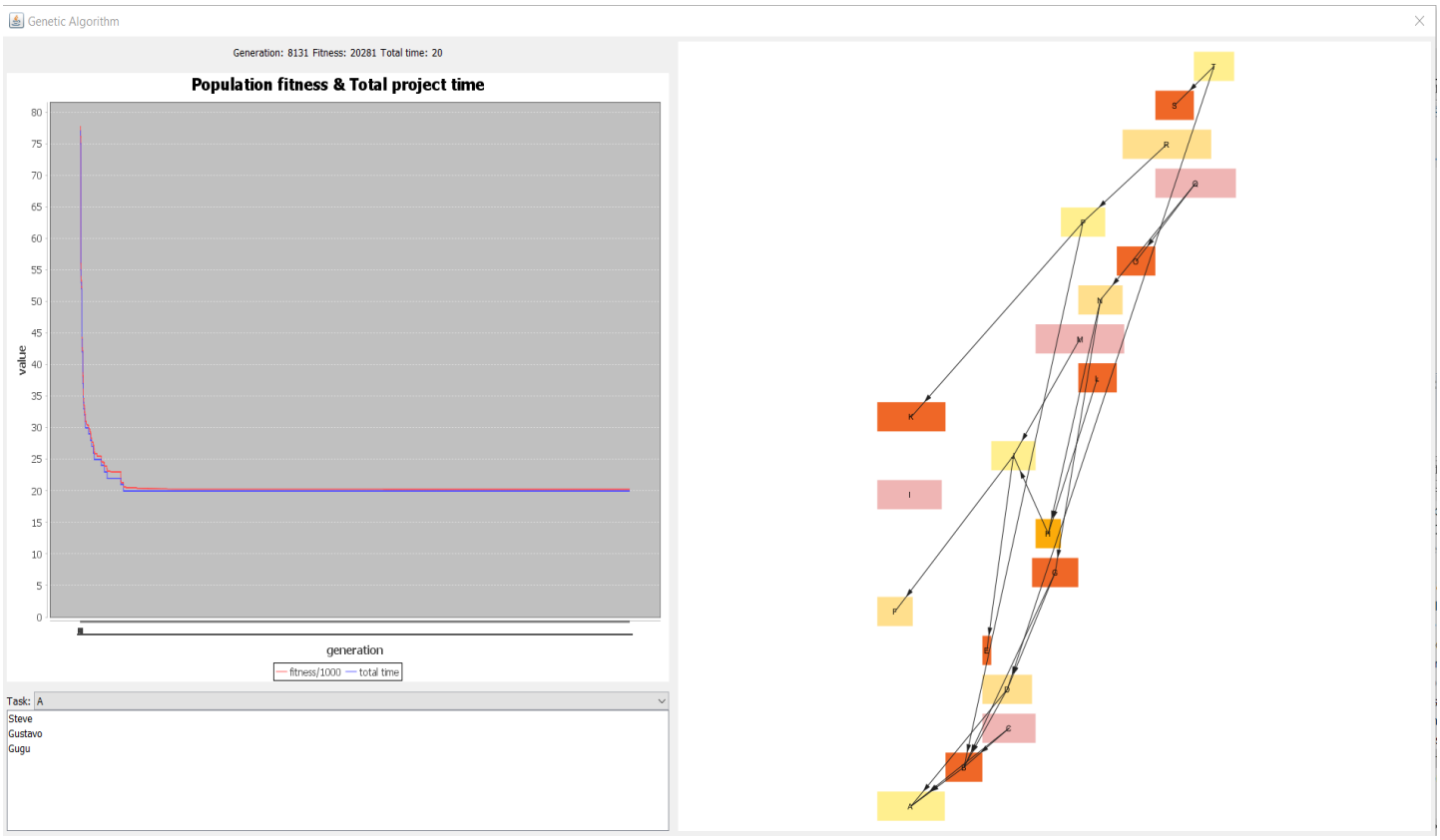

Figura 7: Resultados obtidos.

Do lado direito da figura pode-se observar o grafo de precedências de tarefas. Do lado esquerdo apresenta-se o gráfico que representa a evolução do valor de fitness ao longo das gerações. A figura 8 mostra o gráfico com mais detalhe.

Figura 8: Resultados obtidos.

Arrefecimento Simulado

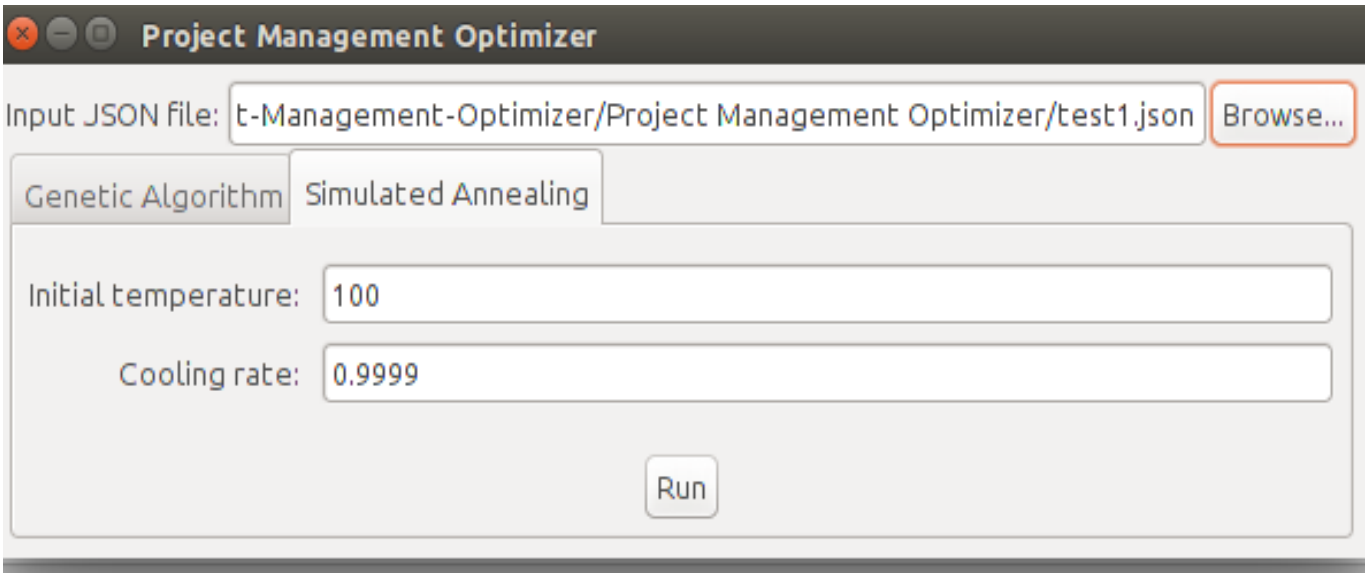

Figura 9: Configurações para o arrefecimento simulado usados no test1.

Os resultados obtidos foram os apresentados na figura 10.

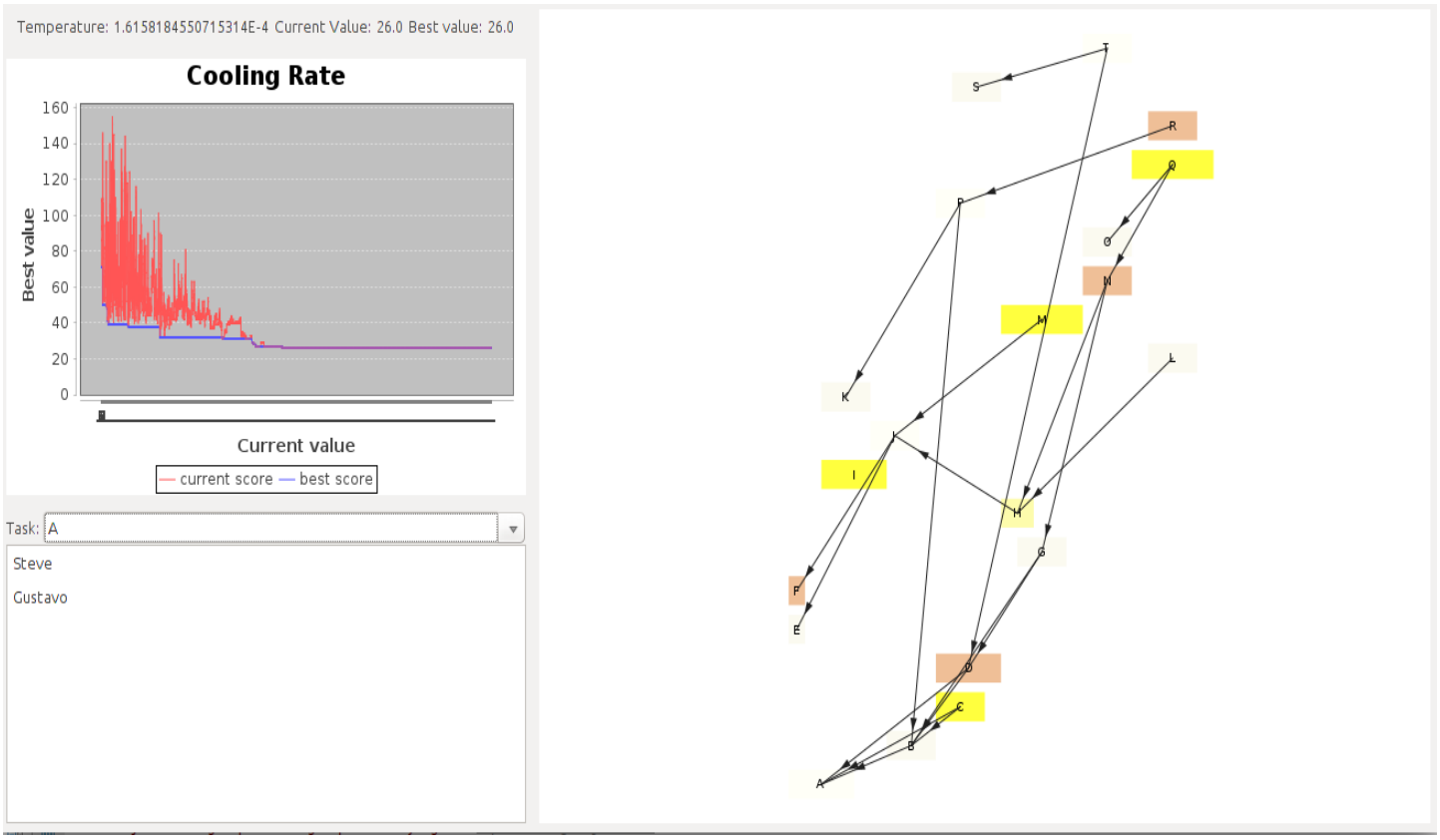

Figura 10: Resultados obtidos.

Do lado direito da figura pode-se observar o grafo de precedências de tarefas. Do lado esquerdo apresenta-se o gráfico que representa a evolução do valor de fitness ao longo das gerações. A figura 11 mostra o gráfico com mais detalhe.

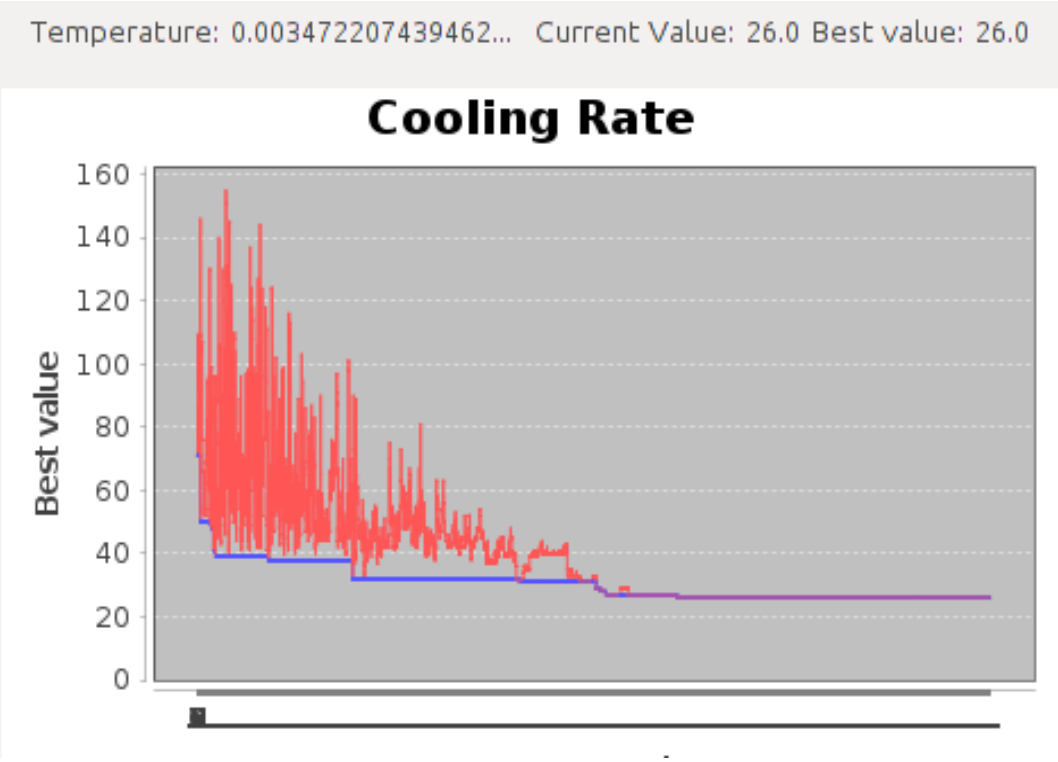

Figura 11: Resultados obtidos.

Resultados

Tabela 1: Resultados combinados do algoritmo genético e do arrefecimento simulado para a mesma experiência.

Algoritmo	Resultados		
Algoritmo Genético	Valor	Fitness/1000	Tempo Total
Tamanho da população	50	19.78	19
Mutation Rate	0.005		
Crossover Rate	0.5		
Elitism	5		
Arrefecimento Simulado	Valor	Fitness/1000	Tempo Total
Temperatura Inicial	100	26	26
Cooling Rate	0.9999		

Conclusões

INSERIR CONCLUSÕES AQUI

Melhoramentos

6.1 Heurística de atribuição de Elementos

A atribuição dos elementos a uma task pode ser melhorada. Em ambos os algorítmos, apenas é necessário que haja pelo menos um elemento disponível com a skill correspondente para que uma task seja alocada. Contudo, pode haver situações em que seja mais vantajoso esperar um certo tempo para que hajam mais elementos a trabalhar na tarefa o que vai diminuir o tempo da mesma.

Para isso seria necessário o estudo da seguinte função procura reflectir o tempo de final da tarefa se procurarmos alocar os membros.

$$f(a,b,\ldots) = \frac{t}{p_a + p_b + \ldots} + g(a,b,\ldots) \tag{6}$$

i - tempo actual

g(a,b,...) - tempo a esperar até que todos os elementos estivessem disponíveis

Fazendo uma análise da função poderíamos utilizar esta heurística para optimizar ambos os algorítmos e chegar assim a soluções que se aproximassem mais do óptimo global

Outros Melhoramentos

Outros melhoramentos a serem implementadas passariam por realizar mais testes por forma a perceber quais as condições que levariam a uma mehor otimização.

Divisão de tarefas

O grupo considera que o trabalho foi desenvolvido de uma maneira saudável e todos os elementos do grupo trabalharam de forma igual (33.3% para cada elemento).

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.