|  |  |
| --- | --- |
| Inteligência Artificial - 3º ano do Mestrado Integrado em Engenharia Informática e Computação | |
| Otimização da Distribuição de Produtos |
| Relatório | |



|  |
| --- |
| Trabalho elaborado por:  Luís Gonçalves – ei12080 – [ei12080@fe.up.pt](mailto:ei12080@fe.up.pt)  Ricardo Loureiro – ei12034 – ei12034@fe.up.pt  Tiago Ferreira – ei12051 – ei12051@fe.up.pt  5/6/2015 |

Objetivo

No âmbito da Unidade Curricular de Inteligência Artificial foi-nos proposta a criação de um algoritmo de otimização para um problema de distribuição de produtos. Com o objetivo de resolver de uma forma eficaz será utilizado um algoritmo genético de modo a minimizar o custo de necessário para satisfazer os diversos pontos de venda.

Especificação

# Análise detalhada do tema

Um fabricante possuí várias Fábricas de produção que têm de abastecer diversos Pontos de Venda a nível nacional. Estas Fábricas têm um limite de produção e encontram-se a uma distância de cada Pontos de Venda. Os pontos de venda requerem mensalmente uma quantidade necessária de produtos tendo de ser satisfeitos pelas fábricas de produção.

O custo de venda é diretamente proporcional à distância entre a Fábrica de produção e o Ponto de Venda, com o objetivo de minimizar este custo pretende-se encontrar para cada ponto de venda as fábricas que os vão fornecer. Será aplicado um algoritmo genético tendo em conta diversos pontos de venda, várias Fábricas de produção, várias distâncias (baseadas na localização das Fábricas e Pontos de Venda) e diferentes capacidades requeridas pelos Pontos de Venda.

# Abordagem

## Algoritmo Adotado

O Algoritmo utilizado neste projeto foi **Algoritmo Genético** que é ideal para encontrar soluções para problemas complexos. Frequentemente usado em áreas como a engenharia por forma a criar produtos próximos dos ideais devido à capacidade de percorrer/pesquisar dentro de uma grande gama de combinações e parâmetros e encontrar o melhor.

Como sabemos estes algoritmos são baseados no processo de seleção natural, isso significa que estes tomam as propriedades fundamentais da seleção natural e aplicam-nas a qualquer problema que estejamos a tentar resolver.

O processo do Algoritmo Genético consiste basicamente num conjunto de passos:

1. Inicialização – Criação de uma população inicial. Esta população é geralmente gerada aleatoriamente e pode ser de qualquer tamanho desejado, desde um pequeno número de indivíduos até milhares.
2. Avaliação - Cada membro da população é então avaliado e calcula-se uma “aptidão” para cada indivíduo. O valor de “fitness” é calculado consoante os nossos requisitos desejados. Estes requisitos podem ser simples, 'algoritmos mais rápidos são melhores', ou mais complexas, 'materiais mais fortes são melhores, mas eles não deve ser muito pesado ". No nosso caso foi tida em conta a distância e a quantidade de produtos a ser distribuídos.
3. Seleção - Queremos constantemente melhorar o condicionamento físico geral das nossas populações. A seleção ajuda-nos a fazer isso, descartando os projetos menos aptos e só mantendo os melhores indivíduos na população. Existem alguns métodos de seleção diferentes, mas a ideia básica é o mesmo, torná-lo mais provável que os indivíduos mais aptos sejam selecionados para a nossa próxima geração.
4. *Crossover* - Durante o “*Crossover*” são criados novos indivíduos, combinando aspetos de indivíduos selecionados. A esperança é que através da combinação de certos traços de dois ou mais indivíduos, vamos criar uma prole mais “*fit*”, que irá idealmente herdar as melhores características de cada um de seus pais.
5. Mutação – É necessário adicionar um pouco de aleatoriedade à genética das nossas populações de outra forma cada combinação de soluções estaria demasiado dependente da geração inicial. A Mutação faz mudanças muito pequenas de forma aleatória para um genoma de um conjunto de indivíduos.
6. Repetir - Agora nós temos a nossa próxima geração podemos começar de novo a partir do passo dois até chegar a uma condição que nos pareça ideal ou até termos atingido um número de repetições “suficientes”.

## Técnicas e Representação do conhecimento

A representação do conhecimento apresentada no Relatório Intermédio foi desde então corrigida tendo em conta as sugestões da Docente da turma e depois de alguns aperfeiçoamentos chegámos à representação final que se encontra, então, no projeto final.

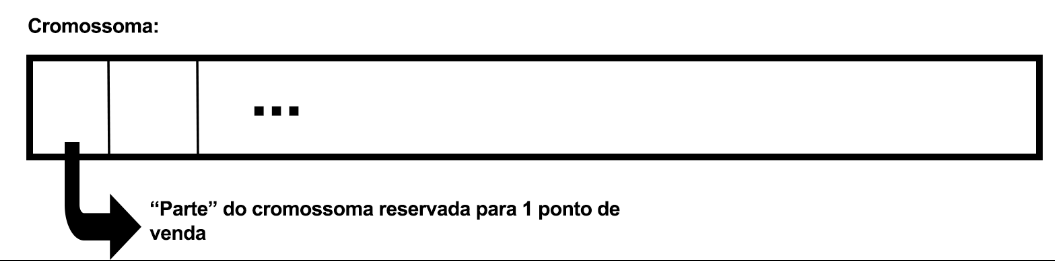
Cada cromossoma é divido pelo número de pontos de venda:

Figure 1: Representação do cromossoma

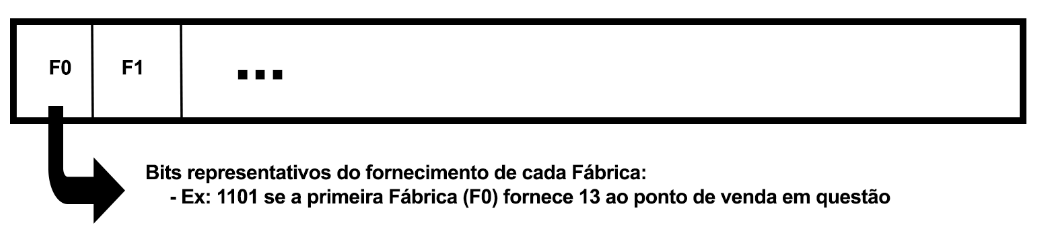
Depois, cada ponto de venda tem um espaço reservado a cada fábrica indicando o que a mesma o fornece:

Figure 2: Representação da parte do cromossoma reservada a um Ponto de Venda

Assim sendo, de forma sucinta, representamos em bits o que cada fábrica forneça cada ponto de serviço por forma a ter toda a informação necessária nos cromossomas mutável. Optámos pela representação binária por forma a facilitar os métodos específicos do Algoritmo Genético, mais especificamente o *crossover* e a mutação.

## Métricas

### Função de Penalização (*Penalty*):

A Função de Penalização tem em conta se alguma das fábricas envia mais do que produz bem como se algum ponto de venda recebe mais ou menos do que o pedido, penalizando mais quanto maior for cada um dos valores.

Penalização(Xᵢ) =

E[Fᵢ] - Excesso de produção de cada Fábrica.

If( Produção < Envio) {

E[Fi] += (Envio - Produção);

}

R[Pᵢ] – O que sobra ou o excesso do recebido por cada Ponto de Venda

R[i] += (Envio - Produção);

ABS(R[i]);

### Função de Adaptação (*Fitness*):

A função de adaptação para quando existe penalização é diretamente proporcional ao inverso do somatório da distância com a penalização.

Adaptação(Xᵢ) =

dᵢ = Distância da fábrica a um ponto de venda, caso esta o forneça.

Se a penalização for nula então a adaptação dependerá só da distância e vice-versa.

Desenvolvimento

# Linguagem e Ambiente

A aplicação foi, na sua totalidade, desenvolvida em java sendo que os *IDEs* utilizados foram o *IntelliJ* e o *Eclipse* *Juno* tanto em *Windows* como em *MacOS*.

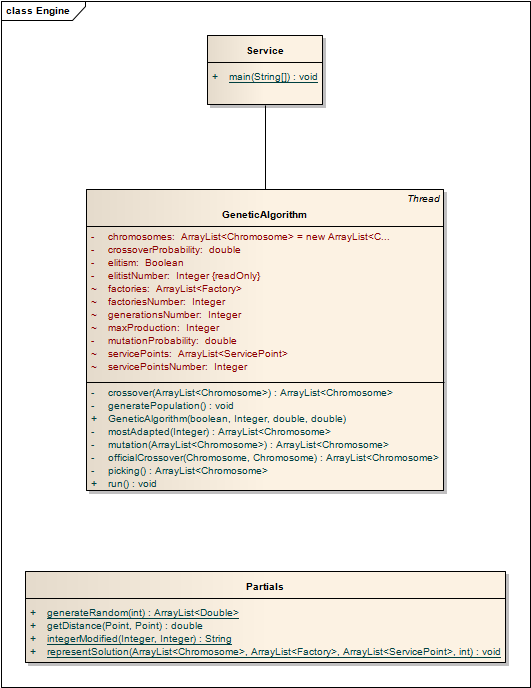
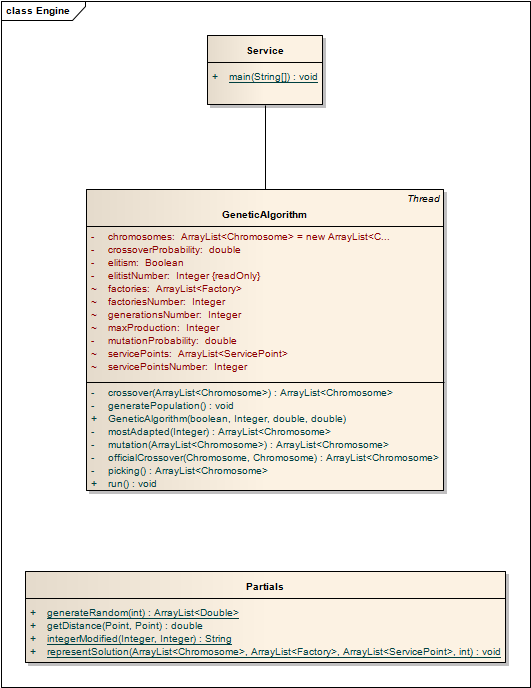
# Estrutura da Aplicação

Por forma a manter o código organizado, dividimo-lo em dois *packages*: o *Engine Package*, que é o responsável pelo funcionamento da aplicação, e o *Models Packages*, em que se encontram as classes representativas consoante o modelo adotado de representação.

No *Engine Package* temos a classe *GeneticAlgorithm* responsável pela execução do algoritmo na sua totalidade, a classe *Service* onde é chamado e inicializado o *GeneticAlgorithm* e inseridos os valores relativos ao mesmo (se é elitista, a probabilidade de *crossover* e a probabilidade de mutação), por fim temos a classe *Partials* usada para cálculos auxiliares e representação da solução final.

No *Models Package* contem a classe *Factory* onde são guardadas as informações relativas à fabrica, a classe *Chromossomes* onde além da informação de cada cromossoma também contem a função de Adaptação e Penalização, e, por fim, a classe *Service Point* que contém toda a informação dos *Service Point*.

# Diagramas de Classes



Experiências

De forma a testar e estudar mais facilmente um caso específico foi desenvolvido um caso-teste bastante simplificado do problema em questão.

Dados do caso-teste são:

* Fábrica:
  + Coordenadas – ( 1 , 1);
  + Produção – 50 unidades.
* Ponto de serviço:
  + Coordenadas – ( 7 , 7);
  + Produção necessária – 50 unidades.

Com estes dados, facilmente seria de esperar uma população final muito próxima da distribuição das 50 unidades necessárias a partir da fábrica em questão. Foi então corrido o algoritmo para 7 cromossomas em cada população, sendo este executado um milhão de vezes, ou seja, 7 cromossomas com 1 000 000 de gerações.

Apresentação da solução:

|  |  |  |
| --- | --- | --- |
|  | População inicial (\*) | População Final(\*) |
| 0 | 7 | 49 |
| 1 | 40 | 49 |
| 2 | 60 | 49 |
| 3 | 43 | 49 |
| 4 | 37 | 49 |
| 5 | 47 | 49 |
| 6 | 39 | 49 |

\* Produção recebida pelo ponto de serviço.

Avaliando os resultados, aconteceu exatamente o esperado, partindo de uma população muito pouco apta à solução indicada, a aplicação do algoritmo um milhão de vezes melhorou muito cada uma das soluções apresentadas. Seria ainda de esperar, que se houvesse um nível de mutação superior (foi usado neste teste uma taxa de 1% de mutação) eventualmente fosse encontrada a solução ótima.

Passemos agora então a um caso mais complexo, de forma a testar com vários pontos de serviço bem como com várias fábricas de produção.

Dados do problema:

* Fábricas:
  + Fábrica 1 :
    - Coordenadas – ( 1 , 1 );
    - Produção – 50 unidades.
  + Fábrica 2 :
    - Coordenadas – ( 7 , 7 );
    - Produção – 60 unidades.
* Pontos de serviço:
  + Ponto de serviço 1:
    - Coordenadas – (2 , 3);
    - Produção requerida – 15 unidades.
  + Ponto de serviço 2:
    - Coordenadas – (4 , 6);
    - Produção requerida – 20 unidades.
  + Ponto de serviço 3:
    - Coordenadas – (5 , 1);
    - Produção requerida – 10 unidades.

Foram usados as mesmas probabilidades do problema anterior, probabilidade de *crossover* de 50% e a probabilidade de mutação de 1%. Foi ainda aproveitado o número de gerações anteriores mas desta vez com 5 cromossomas por população.

Representam-se então de seguida os dados do melhor cromossoma da população inicial:

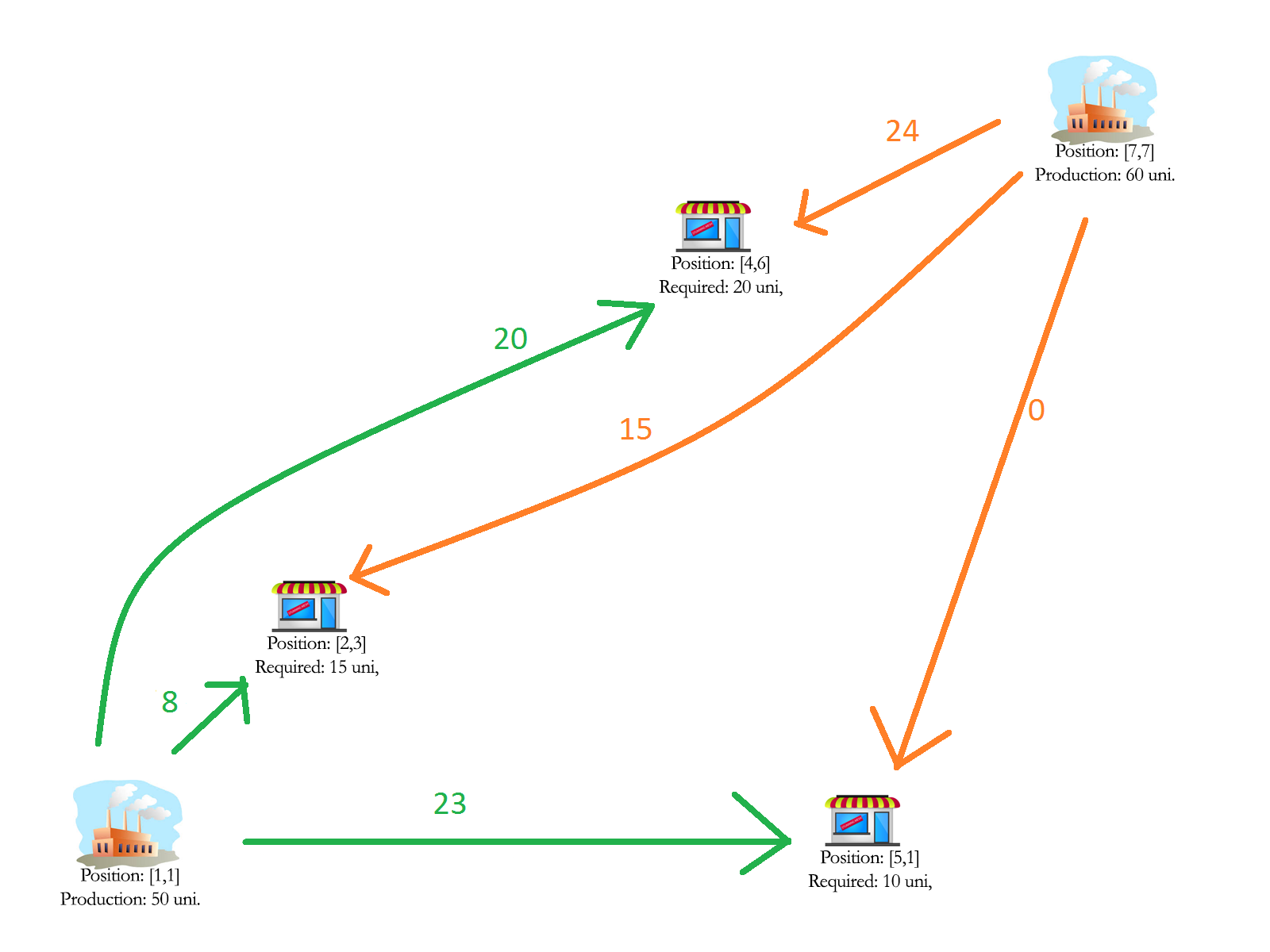
* Cromossoma mais apto da população inicial
  + Ponto de serviço 0
    - 8 da fábrica 0;
    - 15 da fábrica 1.
  + Ponto de serviço 1
    - 20 da fábrica 0;
    - 24 da fábrica 1.
  + Ponto de serviço 2
    - 23 da fábrica 0;
    - 0 da fábrica 1.

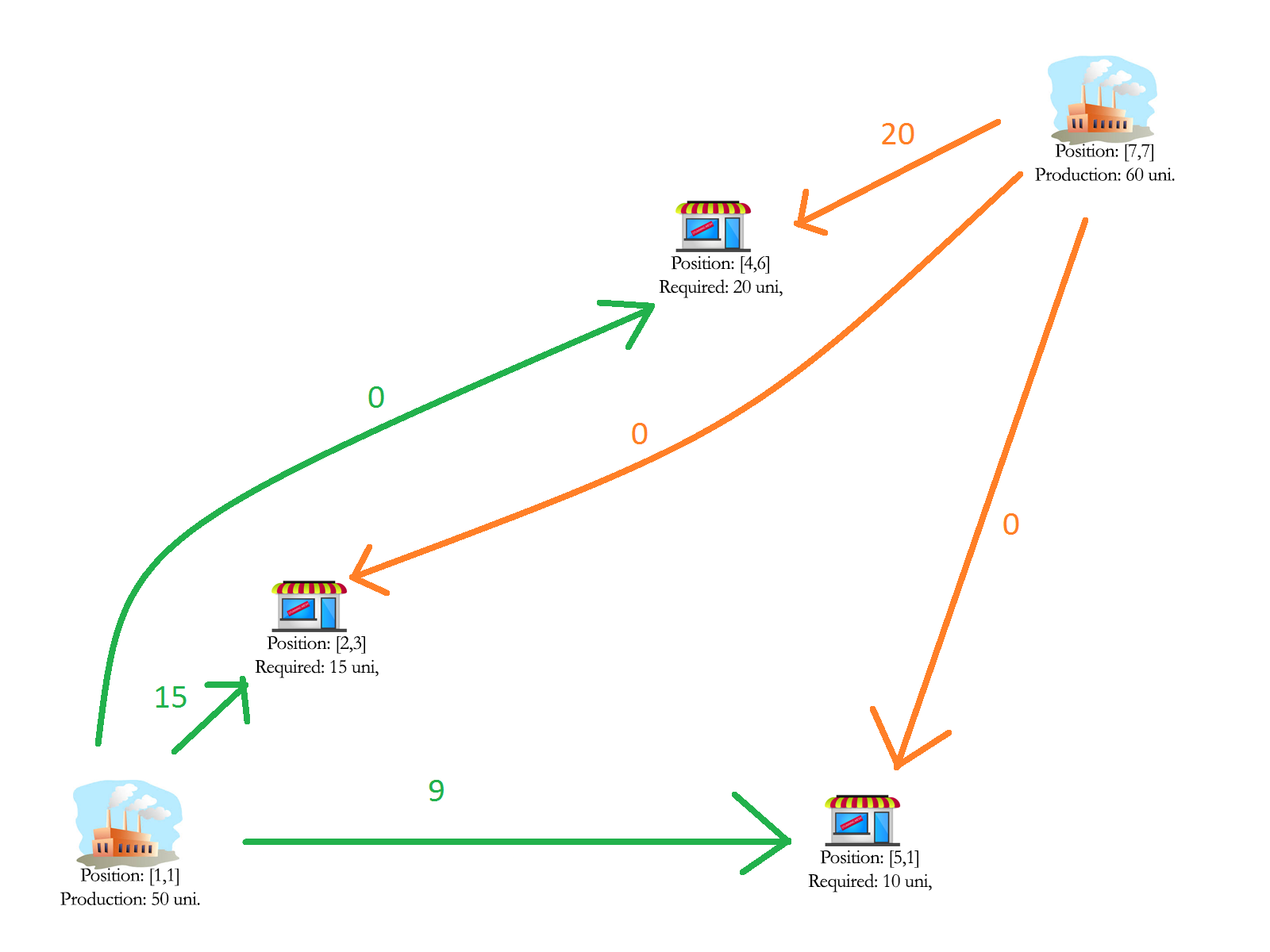
Como podemos ver é facilmente identificável os vários pontos negativos desta solução. Desde já está a receber muito mais de fábricas que não é necessário estar a receber tanto, bem como a fazer viagens de fábricas que não é preciso fazer essas viagens pois uma bastaria para fornecer mais que suficientemente o ponto de serviço.

Foi então aplicado, como dito anteriormente um milhão de iterações, o algoritmo desenvolvido tendo-se chegado a resultados bastante superiores e com elevado nível de adaptabilidade. Será apresentado a seguir da mesma forma o mais apto individuo da população final:

* Cromossoma mais apto da população final
  + Ponto de serviço 0
    - 15 da fábrica 0;
    - 0 da fábrica 1;
  + Ponto de serviço 1
    - 0 da fábrica 0;
    - 20 da fábrica 1.
  + Ponto de serviço 2
    - 9 da fábrica 0;
    - 0 da fábrica 1.

Como podemos ver avaliando este cromossoma, o individuo mais apto da geração final é muito mais apto que o da geração inicial, não só recebendo somente de uma fábrica, o que lhe é necessário, poupando assim várias viagens bem como recebendo, à exceção de uma unidade o ponto de serviço 2, exatamente o que queria.

Segue-se imagens que explicam mais concretamente os dois casos obtidos:



Conclusões

Tendo como base os resultados obtidos nas experiências acima demonstradas, podemos concluir que a implementação do algoritmo genético dado nas aulas teóricas é bastante eficaz. Após um número elevado de gerações, e com a nossa função de avaliação adaptada a um tema específico, podemos observar soluções ótimas.

A referida função de avaliação desenvolvida pelo grupo apresentou, para nós, o maior nível de dificuldade. Esta tinha de ter em conta vários fatores concretos do problema, exigindo, assim, um maior consumo de tempo devido aos vários testes e correções que os seguiram.

Recursos

# Bibliografia

“Métodos de resolução de Problemas e Algoritmos para a Evolução” – www.paginas.fe.up.pt/~eol/1415/ia\_.html, acedido em 6/05/2015.

# Divisão do trabalho

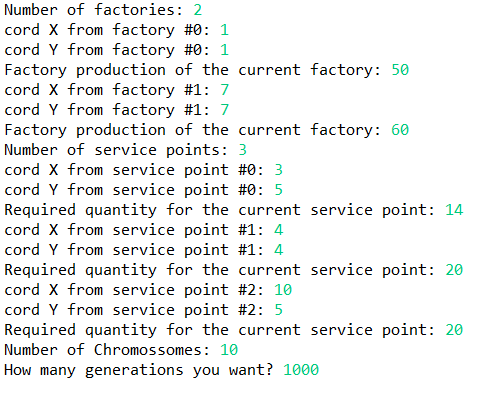
O trabalho foi equitativamente dividido, tendo sido realizado em várias sessões de desenvolvimento por todos os elementos do grupo.

Apêndice

Este projeto não requer a instalação de nenhum software adicional para além de um IDE desenvolvimento com JDK.

Inicialmente ao correr o programa o utilizador é *prompted* com as variáveis pretendidas para o caso específico em estudo.

Segue-se uma *screenshot* do que o programa requer o utilizador fornecer antes deste executar



De seguida começa a execução algoritmo, informando o utilizador inicialmente da população inicial e no fim da sua execução do resultado final da seguinte forma:

