Universidade de São Paulo

IME - Instituto de Matemática e Estatística

MAC0499 - Trabalho de Formatura Supervisionado

Statim

Um Otimizador de rotas

Autores:

Felipe de Godoi Torres

Geraldo Castro Zampoli

São Paulo, SP

2012

Felipe de Godoi Torres
Geraldo Castro Zampoli

Statim

Um Otimizador de rotas

Monografia apresentada junto ao curso de Ciências da Computação da Universidade de São Paulo (USP), na área de exatas, como requisito à obtenção do título de Bacharel.

Orientador:
Prof. Dr. Alfredo Goldman vel Lejbman

São Paulo, SP
2012

Felipe de Godoi Torres
Geraldo Castro Zampoli

Statim

Um Otimizador de rotas

Monografia apresentada junto ao curso de Ciências da Computação da Universidade de São Paulo (USP), na área de exatas, como requisito à obtenção do título de Bacharel.

Banca Examinadora

Prof. Dr. Alfredo Goldman vel Lejbman
Universidade de São Paulo - IME

Prof. Dr. Carlos Eduardo Ferreira
Universidade de São Paulo - IME

São Paulo, 03 de Dezembro de 2012

Aos amigos que nos ajudaram.
Aos professores pelo simples fato de estarem dispostos a ensinar.
Aos orientadores pela paciência e lições aprendidas.

Agradecimentos

Ao Prof. Dr. Alfredo Goldman vel Lejbman pela orientação, amizade e, principalmente, paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Matemática e Estatística pelos seus ensinamentos e aos colegas e amigos de curso que, durante esses anos, nos deram mostra de amizade e companheirismo.

Fairy tales are more than true:
not because they tell us that
dragons exist, but because they
tell us that dragons can be
beaten.

G.K. Chesterson

Resumo

O uso de algoritmos genéticos (AG) – uma classe de algoritmos evolutivos – é uma técnica interessante para encontrar soluções aproximadas em problemas de otimização fazendo uso de processos tipicamente biológicos como *crossing-over*, seleção natural e mutação gênica.

Neste trabalho, apresentaremos o Statim, um otimizador de rotas para entregadores composto por duas partes: a primeira, *mobile*, usa os recursos oferecidos pela plataforma Android para fazer o rastreamento dos aparelhos e a atualização dos itinerários atuais de cada entregador; a segunda, *server*, com os conceitos de AG, usa as informações obtidas dos aparelhos para encontrar a menor rota capaz de atender aos itinerários.

Como existem vários algoritmos diferentes para *crossing-over*, mostraremos os resultados de testes de eficiência em encontrar a rota ótima ao utilizar o *Partially Mapped Crossover* (PMX) e o *Cycle Crossover* (CX). Para os algoritmos de seleção natural, testes semelhantes foram feitos entre *Ranking* e *Wheel*.

Palavras chave: Algoritmos Genéticos, otimização, rotas, PMX, CX, Ranking, Wheel.

Lista de Ilustrações

1 Trajeto do motoboy até o momento. 22
2 Aparece um novo ponto de entrega/coleta. 23
3 O caminho mínimo com esse ponto é encontrado e a rota é refeita. 23
4 Rota R_1. 26
5 Rota R_2. 26
6 *One-point crossover* 28
7 *One-point crossover* com pontos iguais 29
8 *Cycle Crossover (CX)* 29
9 *Cycle Crossover (CX)* - cópia do primeiro ciclo 30
10 *Cycle Crossover (CX)* - cópia trocando os destinos 30
11 *Cycle Crossover (CX)* - resultado 30
12 *Partially Mapped Crossover (PMX)* 31
13 *Partially Mapped Crossover (PMX)* - cópia das extremidades . 31
14 *Partially Mapped Crossover (PMX)* - substituição 32
15 *Partially Mapped Crossover (PMX)* - resultado 32
16 Fluxograma de um algoritmo genético. 42
17 *Strategy Pattern*: Diagrama UML. 43
18 10000 execuções, $p = generation$, $q = 5$ 45
19 5000 execuções, $p = generation$, $q = 10$ 46

Sumário

1 Introdução **11**

1.1 Contextualização 11

1.2 Organização do Trabalho 11

2 Fundamentos **13**

2.1 introdução 13

2.2 A Plataforma Android 14

2.3 Google Directions Service 14

2.4 *Google Geocoding Service* 16

2.5 C2DM: enviando mensagens de um computador para um dispositivo móvel 17

2.6 Algoritmos Genéticos 19

3 Um modelo inicial **22**

3.1 Descrição 22

3.2 Formalizando 23

3.2.1 Hipóteses e condições de contorno 25

4 Uma abordagem genética **26**

4.1 Representação 26

4.2 Geração da População inicial 27

4.3 Operador de *crossover* 28

4.3.1 *One-point crossover* 28

4.3.2 *Multi-point crossover* 31

4.4 Função de aptidão (*fitness*) 33

4.5 Seleção dos mais aptos 34

4.5.1 Seleção por roleta 34

4.6 Critério de parada 35
4.6.1 Limite de gerações 35
4.6.2 Popularidade 36

5 Statim *Mobile* **37**
5.1 Descrição 37
5.2 Arquitetura 37

6 Statim *Server* **39**
6.1 Descrição 39
6.2 Arquitetura 39
6.2.1 Estrutura do banco de dados 39
6.2.2 Estrutura dos *Controllers* 39
6.2.3 Camada *View* 40
6.2.4 JavaScripts que compõem a View 41
6.3 O otimizador 42
6.3.1 Aspectos interessantes da implementação 42

7 Análise dos resultados **45**

8 Parte Subjetiva **48**
8.1 Felipe 48
8.1.1 Desafios e frustrações 48
8.1.2 Disciplinas cursadas mais relevantes 48
8.1.3 Futuro 50
8.1.4 Considerações finais 50
8.2 Geraldo 52
8.2.1 Desafios e frustrações 52
8.2.2 Disciplinas cursadas mais relevantes 52

8.2.3 Futuro 53
8.2.4 Considerações finais 53

9 Conclusão **54**

A *Google Cloud Messaging* **55**

B Timeline **56**

Referências **58**

1 Introdução

1.1 Contextualização

Nos tempos atuais, é muito comum o uso de serviços de entrega e retirada de produtos, acionamos esse serviço quando compramos uma pizza pelo telefone, enviamos uma encomenda via Sedex, solicitamos algum documento com urgência... Em todas essas situações, há uma necessidade de se fazer essas tarefas com a maior eficiência possível, pois, nesse caso, isso impactaria no tempo gasto por viagem, na quantidade de entregas efetuadas no dia e, principalmente, na economia de combustível dos veículos envolvidos, aumentando os lucros das empresas de transporte de encomendas.

Para atender esse nicho, este trabalho consiste numa ferramenta cliente-servidor para fazer o rastreamento dos celulares utlizados pelos transportadores e, com o uso de algoritmos de otimização de rotas, definir o veículo mais apropriado para ser designado a uma determinada localização.

1.2 Organização do Trabalho

Inicialmente, mostraremos os conceitos aprendidos sobre as principais tecnologias abordadas neste trabalho, como a plataforma Android, a API do Google Maps e o C2DM: um serviço de envio de notificações a devices usando o *cloud* do Google.

Após isso, detalharemos os principais aspectos da implementação do Statim Mobile e Server. Por fim, mostraremos o módulo de otimização de rotas do Statim Server.

Nos útimos capítulos, faremos uma análise dos resultados obtidos com esse projeto e diremos nossas impressões pessoais.

2 Fundamentos

2.1 introdução

Não é de hoje que os celulares são os protagonistas de grandes revoluções na forma de se comunicar e consumir informações. Essa revolução ocorre desde os primeiros aparelhos que eram apenas capazes de efetuar chamadas, pois tornou-se possível conversar com pessoas em qualquer lugar sem estar fixo a um computador ou telefone.

Ao longo do tempo, com a diversificação das ferramentas de comunição, a crescente necessidade de conectividade e do rápido consumo de informações, o celular foi se tornando cada vez mais complexo, com inúmeras outras funcionalidades além de simplesmente efetuar chamadas telefônicas. Esses celulares, conhecidos como *smartphones*, atualmente estão tão semelhantes a computadores que é comum ouvirmos expressões do tipo:

"Hoje formatei o meu celular."
"Instalei um programa muito legal no meu celular."
"Este celular tem um processador com dois núcleos."

Para lidar com tamanha complexidade e gerenciar todos os recursos internos do aparelho, uma classe de programas tem ganhado cada vez mais importância também no mundo mobile: os sistemas operacionais.

Entre os grandes nomes que compõem a linha do tempo dos sistemas operacionais para dispositivos móveis, podemos citar: Palm OS, Windows

Mobile, Blackberry OS, iPhone OS, Android, Symbian OS e Windows Phone. Faremos um breve aprofundamento sobre o sistema Android, que usaremos neste trabalho.

2.2 A Plataforma Android

Criado pela *Open Handset Alliance*, uma aliança formada por grandes empresas como Google, HTC, Intel, LG, Motorola, Nvidia, Samsung Electronics com o objetivo de criar novos padrões abertos para tecnologias móveis, a primeira versão beta do Android é datada de 05 de novembro de 2007[1].

A plataforma Android é construída sobre um *kernel* Linux (atualmente na versão 2.6) usando a linguagem Java e, entre outras características, possui sua própria máquina virtual, chamada Dalvik, que compila aplicações Java em *bytecode* permitindo, assim, que aplicações sejam escritas em Java e distribuídas em bytecode para outros sistemas Android. No entanto, como essa máquina virtual não executa *bytecode* Java, não podemos considerá-la uma JVM[2].

2.3 Google Directions Service

Desenvolvido pelo Google para aumentar as funcionalidades do *Google Maps* e fornecer uma poderosa ferramenta para o cálculo de rotas, o *Google Directions* é um serviço que, por meio de uma requisição HTTP ao *Google Maps API Directions Service*, permite a exibição de uma rota ligando dois pontos quaisquer do mapa, usando JavaScript.

Além de simplesmente mostrar o caminho mais rápido, há uma série de parâmetros que podem ser personalizados, como:

- tempo estimado da viagem a pé, usando um automóvel ou bicicleta;
- cálculo de rotas alternativas;
- determinar rotas evitando rodovias ou pedágios;
- exibir caminhos passando por determinados pontos pré-definidos: os *waypoints*.

O principal objeto fornecido pelo *Google Directions* é o `DirectionsService`, pois é por meio dele que fazemos uma requisição HTTP e recebemos a resposta em formato JSON. Após isso, criamos um `DirectionsRenderer`, que faz o *parsing* do JSON recebido e, com os elementos `LatLng` encontrados, exibe a rota no mapa por meio da função `setDirections`.

Veja um exemplo em JavaScript:

```
function calculateRoute(origin, destination, map) {
    var service = new google.maps.DirectionsService();
    var renderer = new google.maps.DirectionsRenderer();
    renderer.setMap(map);
    var request = {
            origin: origin,
            destination: destination,
            };
    service.route(request, function(response, status) {
        if (status == google.maps.DirectionsStatus.OK)
            renderer.setDirections(response);
    });
}
```

Neste trabalho, usaremos esse serviço para calcularmos as rotas entre os diversos devices rastreados exibidos no mapa e um ponto de destino.

2.4 *Google Geocoding Service*

Com a intensa utilização do *Google Directions Service*, surgiu a necessidade de se manipular coordenadas no mapa de forma mais legível, pois, a menos que tenhamos muita habilidade na leitura de latitudes e longitudes, é difícil dizer, por exemplo, a qual localidade o ponto (-23.55937, -46.731833) se refere.

Para resolver esse problema, os engenheiros do Google criaram um serviço chamado *Google Geocoding*, que tem como objetivo fazer o mapeamento entre uma coordenada (latitude, longitude) e o seu respectivo endereço.

Assim como o *Directions Service*, este serviço também faz uma requisição HTTP e retorna um JSON com muitas informações, dentre as quais está o endereço *human-readable*.

Além disso, ao se passar um endereço legível para este serviço, é efetuado o *reverse geocoding*, que tem como resposta a coordenada referente ao endereço passado previamente.

No trabalho desenvolvido, usaremos o *Google Geocoding* para permitir que o ponto de destino das rotas dos entregadores seja inserido pelo usuário sem o uso de algum conversor, ficando a cargo do nosso software essa conversão.

Veja um exemplo:

```
function plotAtAddress(address) {
    var geocoder = new google.maps.Geocoder();
        geocoder.geocode({'address': address},
                         function(results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
                var coordinate = results[0].geometry.location;
                // plot at coordinate.
            }
        });
}
```

2.5 C2DM: enviando mensagens de um computador para um dispositivo móvel

Para a comunicação entre um computador e um dispositivo móvel, existe um serviço do Google chamado *Cloud to Device Messaging (C2DM)*, que permite enviar pequenas mensagens a devices previamente cadastrados.

O C2DM não tem como objetivo o tráfego de grandes informações pela rede, pelo contrário, na página dos desenvolvedores Android, há uma forte recomendação de que esse serviço seja usado apenas para enviar sinais de notificação, e, então, se for o caso, o tráfego de dados "pesado" seria feito pelo próprio *device*.

Esse serviço possui a limitação de 200.000 mensagens por dia por en-

viador (*sender*) que, segundo a página do C2DM, é suficiente para a maioria das aplicações de pequeno e médio porte. Para uma cota maior deve-se enviar uma mensagem para os mantenedores do serviço explicando o motivo.

Uma aplicação usuária desse serviço deve satisfazer duas condições principais:

- O tamanho da mensagem enviada deve se limitar a 1024 bytes. Caso seja enviada uma mensagem acima desse limite, uma exception será gerada.
- Como o C2DM não garante a ordem das mensagens enviadas, a aplicação não deve depender da sequência de envio das mensagens.

O mais interessante desse serviço é que o *device* não precisa estar rodando a aplicação que usará a notificação do C2DM. A informação chegada dispara um *Broadcast* interno que será o responsável por "acordar" nossa aplicação, responsável por tratar a notificação enviada pelo C2DM. Para isso, basta que tenhamos classes filhas de BroadcastReceiver que esperem Intents com *actions* do C2DM.

Para criarmos uma aplicação que utilize o C2DM, temos que entender, primeiramente, sua dinâmica. Para uma mensagem ser enviada, ela passará por três lugares:

- Aplicação servidora: ela pode ser programada em qualquer linguagem e será responsável por enviar um POST, com a nossa mensagem, para os servidores do Google.

- *Cloud* C2DM do Google: esse *cloud* será o responsável por receber o POST do nosso servidor e enviar uma mensagem a todos os *devices* previamente cadastrados.

- Aplicação *mobile*: nossa aplicação *mobile* deve estar preparada para receber notificações *Broadcast*. Dessa forma, ela não precisará estar ativa para tratar a mensagem recebida pelo C2DM.

2.6 Algoritmos Genéticos

A idéia dos algoritmos genéticos (ou GA, do inglês, *Genetic Algorithms*) foi concebida por John Holland nos anos 60, inicialmente com a intenção de simplesmente estudar como os organismos se reproduzem na natureza. Em paralelo, estudos sobre computação evolucionária (*Evolutionary Computation*) estavam sendo feitos por vários cientistas da computação.

Mais tarde, as idéias de Holland foram aprimoradas, introduziram-se conceitos de mutação genética, populações, cruzamentos e inversão de genes, até chegar ao modelo atual dos GAs, hoje uma área de pesquisa de grande importância da Inteligência Artificial.

Em uma definição breve, os algoritmos genéticos são um método para programar a busca das melhores soluções de um problema dada uma coleção de soluções candidatas.

Se estamos resolvendo um problema, geralmente procuramos por alguma solução que seja melhor que outras. O espaço de todas as soluções possíveis (que é um conjunto entre as quais a solução procurada está) é cha-

mado Espaço das Soluções (ou Espaço de Estados). Cada ponto desse estado representa uma solução possível. Cada uma das possíveis soluções pode ser "marcada" pelo seu valor (ou adequação) para o problema. Com os Algoritmos Genéticos nós procuramos pela melhor solução dentre um número de possíveis soluções - representadas por pontos no Espaço de Soluções. [4] [5]

Para calculamos a quanto uma solução está perto de ser a solução ótima fazemos uso de uma função $f(x)$, chamada função de *fitness*.

Como em nosso sistema teremos um problema de otimização, o uso de algoritmos genéticos nos pareceu apropriado, para tanto fizemos alguns testes posteriores implementando alguns algoritmos e testando seus desempenhos, a implementação de uma algoritmo genético segue mais ou menos esses passos:

1. **Início:** Gere uma população aleatória de n cromossomos (soluções adequadas para o problema)
2. **Adequação:** Avalie a adequação $f(x)$ de cada cromossomo x da população
3. **Nova população:** Crie uma nova população repetindo os passos seguintes até que a nova população esteja completa
   - 3.1 **Seleção:** Selecione de acordo com sua adequação (melhor adequação, mais chances de ser selecionado) dois cromossomos para serem os pais
   - 3.2 **Cruzamento:** Com a probabilidade de cruzamento cruze os pais para formar a nova geração. Se não realizar cruzamento, a nova geração será uma cópia exata dos pais.
   - 3.3 **Mutação:** Com a probabilidade de mutação, altere os cromossomos da nova geração nos locus (posição nos cromossomos).

3.4 **Aceitação:** Coloque a nova descendência na nova população

4. **Substitua:** Utilize a nova população gerada para a próxima rodada do algoritmo

5. **Teste:** Se a condição final foi atingida, pare, e retorne a melhor solução da população atual

6. **Repita:** Vá para o passo 2 [5]

3 Um modelo inicial

3.1 Descrição

Suponha um conjunto de entregadores, cada um com o seu respectivo itinerário (figura 1). É bastante comum que, ao decorrer do dia de trabalho, apareçam novos pontos de destino (entregas ou retiradas de documentos, por exemplo) que não foram previstos nos itinerários iniciais (figura 2). Como atribuir esse novo destino a um entregador, para que haja um menor desvio da rota original (figura 3)?

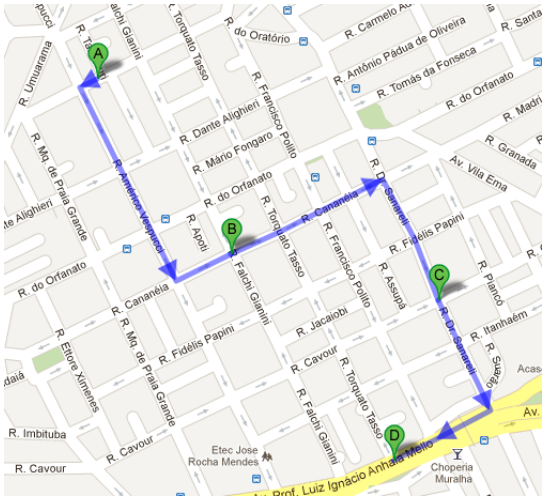


Figura 1: Trajeto do motoboy até o momento.

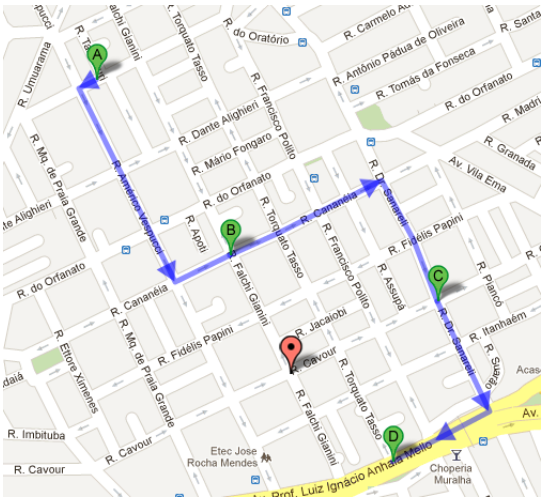


Figura 2: Aparece um novo ponto de entrega/coleta.

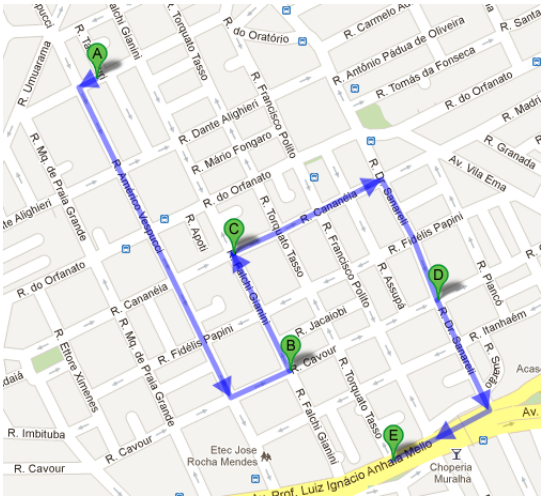


Figura 3: O caminho mínimo com esse ponto é encontrado e a rota é refeita.

3.2 Formalizando

Considere o conjunto $C = \{I_1, I_2, ..., I_m\}$ onde $I_i = (P_{i1}, P_{i2}, ..., P_{iN})$ é uma n-úpla ordenada de pontos (x, y) em $\mathbb{R}^2$.

Definamos:

Um ponto P é da forma (x, y) com $x, y \in \mathbb{R}$.

(a) $d_{a,b}$ = distância[1], não-euclideana, entre os pontos a e b.

Seja P^* um novo ponto a ser inserido em alguma posição k de um itinerário I de tamanho n. Então, usando **(a)**, temos:

(b) $d_{I_i} = \sum_{j=1}^{j<n} d_{P_{ij},P_{ij+1}}$: distância total do itinerário I_i.

(c) $d_{I_i}[P^*]_k = d_{I_i} + [d_{p_{ik-1},P_k^*} + d_{P_k^*,p_{ik+1}}] - d_{p_{ik-1},p_{ik+1}}$: distância total de I_i com P^* na posição k.

Queremos que essa posição k seja tal que, fixando-se o itinerário, ao inserir o novo ponto P^*, nos dê a menor distância total. Sendo assim, usando **(c)**, temos:

(i) $d_{I_i}[P^*]_k \leq d_{I_i}[P^*]_h \forall k \neq h$, I_i fixo

Após obtermos todos os itinerários satisfazendo **(i)**, queremos aquele que tiver a menor distância total, ou seja:

(ii) $d_{I_i}[P^*]_k \leq d_{I_j}[P^*]_h \forall j \neq i$; h, k fixos

Dessa forma, teremos o itinerário procurado.

[1]Como será necessário sabermos a distância real (não-euclideana) entre dois pontos no mapa, faremos uso do Google Maps para obtermos tais informações.

3.2.1 Hipóteses e condições de contorno

(1) Por sair do escopo deste trabalho, consideraremos ótimas as rotas (itinerários) inicialmente atribuídas aos entregadores.

(2) Ao otimizar uma rota, não consideraremos o trânsito, dada a dificuldade em se obter esse tipo de informação para todas as vias (alguns serviços de monitoramento de trânsito do Google só funcionam em vias de grande movimento).

(3) Faremos a otimização das rotas para apenas um ponto novo, pois, caso fizéssemos para um número arbitrário de pontos, teríamos condições de gerar rotas iniciais ótimas (contradizendo **(1)**).

4 Uma abordagem genética

4.1 Representação

Vimos anteriormente que precisamos de uma representação apropriada (que chamaremos de cromossomo) dos elementos que compõem nosso espaço de soluções. Como nosso espaço de soluções é composto por conjuntos de rotas, para um conjunto contendo $i \in \mathbb{N}$ rotas (figuras 4 e 5), adotaremos o seguinte cromossomo:

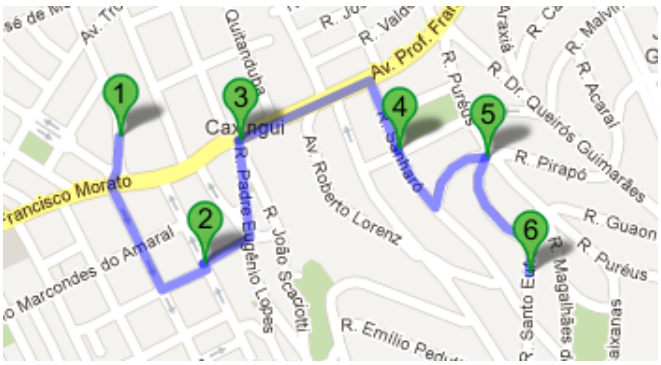


Figura 4: Rota R_1.

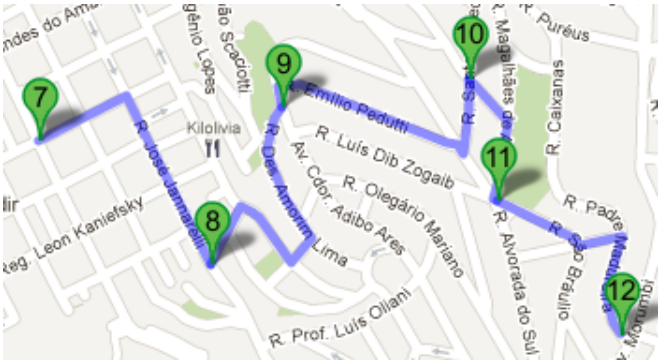


Figura 5: Rota R_2.

$$C = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)$$

Para que possamos ter, de fato, um mapeamento entre cromossomos e rotas, essa representação deve ser capaz de nos dizer quais são as rotas. Da forma que está, não conseguimos extrair essa informação do cromossomo C, pois não há garantias de que as rotas otimizadas terão o mesmo tamanho das rotas iniciais. Sendo assim, precisamos melhorar um pouco a nossa representação.

Faremos uma pequena mudança em C: marcaremos o primeiro ponto de cada rota como um gene especial:

$$C = (\mathbf{1_1}, 2, 3, 4, 5, 6, \mathbf{7_2}, 8, 9, 10, 11, 12)$$

Note que, agora, dado um cromossomo qualquer, as rotas que o compõe são dadas pelas partições iniciadas no gene especial e terminadas no gene da posição anterior ao próximo especial:

$$R_1 = (\mathbf{1_1}, 2, 3, 4, 5, 6)$$
$$R_2 = (\mathbf{7_2}, 8, 9, 10, 11, 12)$$

4.2 Geração da População inicial

Usando a ideia acima, a geração da população inicial é feita trivialmente, pois basta criarmos os cromossomos aleatoriamente com valores inteiros limitados no total de pontos a serem tratados. Usando os pontos das figuras 4 e 5, nossa população inicial seria composta por N cromossomos de tamanho 12, com pontos aleatórios em $[1, 12]$.

4.3 Operador de *crossover*

Crossing-over (ou cruzamento cromossômico) é o nome dado ao processo de geração de cromossomos-filhos por meio da troca de material genético entre dois cromossomos-pais. Devido a essa recombinação, o *crossing-over* é o maior responsável pela diversidade genética. Já, no universo dos Algoritmos Genéticos, a diversidade genética é necessária para explorarmos a maior área possível do nosso espaço de soluções evitando, assim, cairmos nas armadilhas dos ótimos locais.

Existem duas principais classes de operadores, cuja classificação é dada pela quantidade de partes recombinantes:

4.3.1 *One-point crossover*

Essa classe de operadores é caracterizada por cortar os cromossomos-pais em um único ponto e recombiná-los usando diversas estratégias. Estudamos duas técnicas nessa classe: um operador canônico e o *Cycle Crossover (CX)*[9].

- **Operador canônico:** Os dois cromossomos-pais são "cortados" no mesmo ponto e cada cromossomo-filho é gerado com uma parte de cada pai. Como na figura 6.

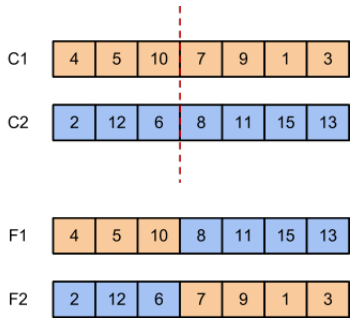


Figura 6: *One-point crossover*

Apesar de ser facilmente implementável, esse operador não se mostra muito eficiente quando queremos que os filhos não apresentem pontos iguais [3]. Veja na figura 7.

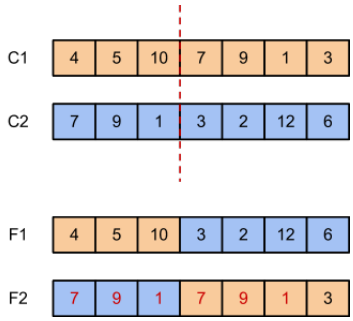


Figura 7: *One-point crossover* com pontos iguais

Note que, apesar de os cromossomos-pais serem válidos (pois não apresentam duplicidade de pontos), o cromossomo-filho $F2 = (7, 9, 1, 7, 9, 1, 3)$ representa uma rota inválida, pois contém duplicidade dos pontos $7, 9, 1$.

- ***Cycle crossover (CX)*:** Um ponto de corte é aleatoriamente selecionado nos cromossomos-pais e o cruzamento segue o algoritmo abaixo:

  (i) Cada filho, F1 e F2, recebe o elemento marcado pelo corte (figura 8).

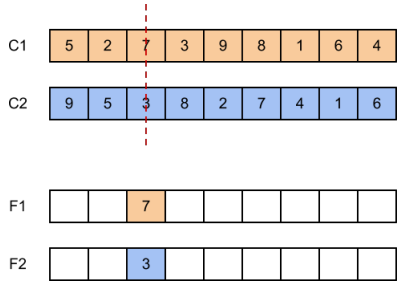


Figura 8: *Cycle Crossover (CX)*

(ii) Caso haja um ciclo contendo os elementos incialmente marcados (figura 9), eles são copiados para os filhos na mesma posição que foram encontrados nos pais.

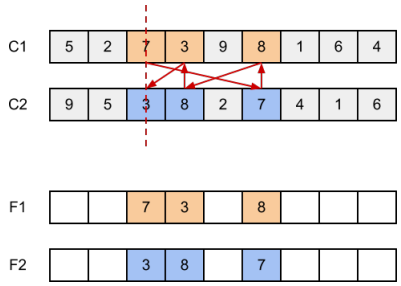


Figura 9: *Cycle Crossover (CX)* - cópia do primeiro ciclo

(iii) Ao término da primeira cópia, outro ciclo é procurado, mas, dessa vez, a cópia é feita trocando-se os filhos, como na figura 10.

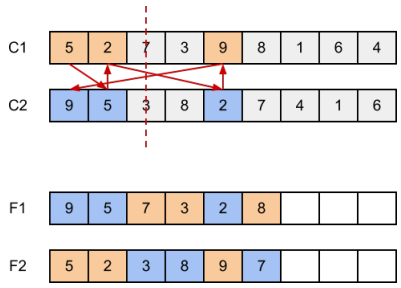


Figura 10: *Cycle Crossover (CX)* - cópia trocando os destinos

(iv) Assim que todos os ciclos tenham sido encontrados e copiados aos filhos, obtemos F1 e F2, de acordo com a figura 11.

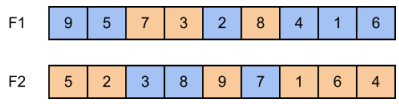


Figura 11: *Cycle Crossover (CX)* - resultado

4.3.2 *Multi-point crossover*

Nesse tipo de operador, os cromossomos-pais são "cortados" em mais de um ponto e recombinados usando diversas técnicas. Descreveremos em detalhes a técnica chamada de *Partially Mapped Crossover (PMX)*[8][10].

- ***Partially Mapped Crossover (PMX)*:** Esse método de recombinação é descrito pelo seguinte algoritmo:

  (i) Dois cromossomos-pais, C1 e C2, são selecionados, dois pontos de "corte" são determinados aleatoriamente e cada filho, F1 e F2, recebe os elementos que estiverem entre esses pontos (figura 12).

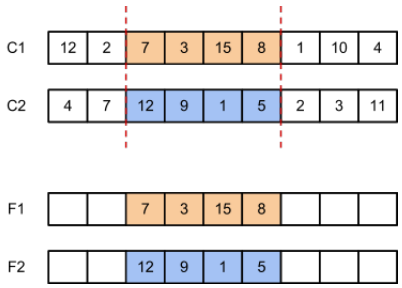


Figura 12: *Partially Mapped Crossover (PMX)*

  (ii) Após isso, copia-se os elementos das extremidades de cada cromossomo-pai, caso não exista nos filhos (figura 13).

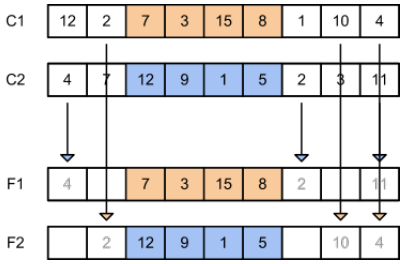


Figura 13: *Partially Mapped Crossover (PMX)* - cópia das extremidades

(iii) Caso o elemento já exista nos filhos, troca-se o elemento a ser copiado por aquele que estiver na posição correspondente à seção de corte do outro pai (figura 14).

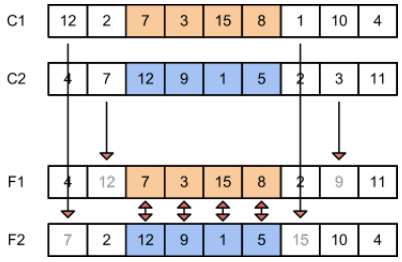


Figura 14: *Partially Mapped Crossover (PMX)* - substituição

(iv) Obtendo, ao final (figura 15), os dois filhos de tal forma que não haja elementos repetidos, garantindo, assim, sua viabilidade como soluções do nosso problema.

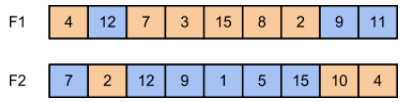


Figura 15: *Partially Mapped Crossover (PMX)* - resultado

Perceba que esse método não gera cromossomos com elementos repetidos. De fato, ao analisarmos o passo (iii) do algoritmo, veremos que um elemento X só será copiado de forma repetida em dois casos:

- **Caso 1:** O mesmo elemento está presente nas mesmas posições dos dois trechos escolhidos em (i) - pois dessa forma, o elemento a ser copiado seria trocado por ele mesmo.

  No entanto, para isso ocorrer, significa que um dos pais já possuia ao menos dois elementos X em posições diferentes (na região escolhida em (i) e na posição em (iii)). Dessa forma, esse cromossomo não seria vá-

lido para gerar novos descendentes.

- **Caso 2:** Há elementos na seção de corte tais que eles formem um ciclo (p.ex.: $C_1[2] = C_2[3] = \text{X}$, $C_1[3] = C_2[2] = \text{Y}$) e, fora da seção de corte de um desses pais (C_1 ou C_2) haja um desses elementos. Dessa forma, o cromossomo-filho possuiria elementos duplicados.

  Novamente, um cromossomo-pai que satisfaça a condição acima descrita é inválido, pois possuirá elementos duplicados.

Portanto, partindo de pais válidos o PMX gerará filhos também válidos.

4.4 Função de aptidão (*fitness*)

A função de aptidão é responsável pela avaliação da sobrevivência do cromossomo ao longo das iterações do Algoritmo Genético e tem papel decisivo na procura pela solução ótima. Como queremos encontrar o itinerário com a menor distância total (como definido na Modelagem, em (ii)), nossa função de aptidão, $f(x)$, será:

$$f(x_i) = \frac{1}{\sum_{j=1}^{j<N} d_{i_j,i_{j+1}}}$$

Para $x_i = (i_1, i_2, i_3, i_4, i_5, i_6, \ldots, i_N), i_1, i_2, \ldots, i_N \in \mathbb{N}$. Caso i_{j+1} represente o início de uma rota, $d_{i_j,i_{j+1}} = 0$

Dessa forma, quanto menor o valor do somatório, mais apto será o cromossomo.

4.5 Seleção dos mais aptos

A criação dos cromossomos-filhos depende, além do operador de crossover, da seleção dos pais aptos para gerar descendentes. No entanto, em alguns casos, apenas capturarmos os cromossomos mais aptos não é uma boa estratégia.

O método de seleção está intimamente ligado à rapidez de convergência do AG[2] a uma solução ótima (note que, ao dizermos "solução ótima", não especificamos se é local ou global); sendo assim, estratégias que potencializam a velocidade de convergência podem ser mais sensíveis aos ótimos locais, ocasionando erros de otimização. No entanto, uma baixa velocidade de convergência implica em mais tempo gasto para encontrar uma solução ótima, inviabilizando, em alguns casos, o uso de um AG.

Existem muitas formas interessantes de seleção dos cromossomos mais aptos. Para este trabalho, usaremos o método da roleta.

4.5.1 Seleção por roleta

Usando a função de aptidão aplicada a cada cromossomo da população, esse método atribui uma chance de seleção ao cromossomo proporcional à sua aptidão, de forma que os mais aptos tenham uma chance maior de serem selecionados, possibilitando também que aqueles menos aptos sejam selecionados.

Definamos o tamanho da roleta por:

[2] Algoritmo genético.

$$W = \sum_{i=1}^{N} f_i$$

Dessa forma, a chance de um cromossomo C_i ser selecionado usando esse método será:

$$W_i = \frac{f_i}{W}$$

Onde f_i é a função de aptidão aplicada a cada cromossomo da população $(C_1, C_2, C_3, \ldots, C_N)$.

4.6 Critério de parada

Estudamos as duas formas comuns de parada num AG: limite de gerações e popularidade do cromossomo.

4.6.1 Limite de gerações

Este critério é bastante simples, porém mais empírico e suscetível a erros de ótimos locais. Sua função é terminar o algoritmo em alguma geração previamente conhecida.

Como cada problema tem suas próprias características de parada, definir uma geração limite genericamente é algo bastante ineficiente, pois pode-se errar tanto ao finalizar o algoritmo tardiamente (demorando mais do que o necessário), como precocemente (causando erros de ótimos locais).

4.6.2 Popularidade

A parada por popularidade é a que apresenta os resultados mais uniformes, pois nesse critério, o AG para quando uma certa porcentagem da população é composta pelo mesmo cromossomo, indicando que a população está convergindo para esse indivíduo que, provavelmente, é o ótimo procurado.

Em comparação com o primeiro critério de parada, seu término é mais lento, no caso geral, mas apresenta uma maior precisão ao retornar uma solução ótima.

5 Statim *Mobile*

5.1 Descrição

O Statim Mobile é o módulo cliente para Android, do Statim. Seu objetivo primário é a determinação e o envio do posicionamento do aparelho no qual ele estiver instalado. Para isso, usamos o GPS ou alguma rede ativa no device, como Wi-Fi ou 3G.

Além disso, esse módulo faz o cadastro do aparelho no C2DM e no banco de dados do Statim Server, para que ele possa receber notificações do serviço do Google.

5.2 Arquitetura

Esse projeto é composto principalmente por quatro classes independentes:

`C2DM_Activity` Única `Activity` do projeto, é inicializada após a sua instalação e tem como objetivo enviar um *Broadcast* interno para o registro desse *device* ao C2DM vinculado a um e-mail previamente cadastrado como *sender*.

`C2DM_RegistrationReceiver` É uma filha de `BroadcastReceiver`, o que significa que essa classe só é chamada quando o *device* emite algum Broadcast interno.

Se o *Broadcast* interno tiver como ação *com.google.android.c2dm.intent.REGISTRATION*, significa que a classe `C2DM_Activity` foi executada com sucesso e que já temos o *id* desse *device* (armazenado em registration_id). Sendo assim, fazemos algumas verificações com esse *id* para sabermos se é um novo valor, ou se

ele já havia sido gerado anteriormente e, por fim, enviamos ao nosso servidor.

`C2DM_MessageReceiver` Também é um `BroadcastReceiver` que tem como objetivo o tratamento das mensagens enviadas pelo sender ao *device*.
Se a ação do *Broadcast* gerado por *com.google.android.c2dm.intent.RECEIVE*, sabemos que chegou uma mensagem ao device, logo, extraímos o conteúdo da chave data.message e, se for *gps*, iniciamos o processo de rastreamento do aparelho usando, inicialmente, seu GPS e, caso ele esteja desligado, alguma rede móvel ou Wi-Fi.

`C2DM_CoordinateSender` A principal diferença entre uma `Activity` e um `Service` é que este permite ser rodado em *background*, sem que a tela do aparelho esteja desbloqueada ou ligada. Como o rastreamento do *device* será feito, em muitos casos, sem que o usuário saiba, o envio das coordenadas não deve depender do aparelho estar desbloqueado.
Sendo assim, criamos essa classe com o objetivo de enviar ao nosso servidor as coordenadas obtidas pelo `C2DM_MessageReceiver`.

6 Statim *Server*

6.1 Descrição

O Statim Server é o responsável por gerenciar os *devices* cadastrados, e, ao receber uma ordem de *ping*, busca pelos aparelhos associados àquele *sender* e envia uma notificação para eles. Além disso, este módulo recebe as coordenadas dos aparelhos notificados e as armazena em memória no formato JSON para que sejam facilmente exibidas pelo mapa, mostrando, assim, suas localizações espaciais.

6.2 Arquitetura

Este projeto está organizado de acordo com a estrutura MVC (*Model, View, Controller*) e o principal *framework* MVC utilizado é o VRaptor3. Para abstraírmos o tipo de banco de dados, utilizamos o *framework* ORM (*object-relational mapping*) Hibernate3. Para a View, utilizamos Javascript em conjunto com a API do *Google Maps*, para a exibição de informações referentes à localização dos aparelhos rastreados.

6.2.1 Estrutura do banco de dados

O nosso banco de dados contém duas entidades: `Device` e `Sender` que representam, respectivamente, os aparelhos restreados e aqueles que têm acesso à visualização das informações referentes ao monitoramento desses aparelhos. Como um `Sender` pode ter muitos `Devices`, e um `Device` pertence a apenas um `Sender`, a relação entre `Sender` e `Device` é *One-to-Many*.

6.2.2 Estrutura dos *Controllers*

Na camada *Controller*, temos quatro classes notáveis:

`DeviceController` Responsável por salvar e atualizar os ids de registro, enviados pelo C2DM, dos aparelhos móveis.

`SenderController` Sua única função é permitir o cadastro dos enviadores.

`PingController` Tem como objetivo encontrar os *devices* de um determinado *sender* e enviar uma notificação a eles.

`MapController` Gerencia em memória as coordenadas enviadas pelos *devices* após uma notificação e alimenta um JSON com essas informações para que possam ser consumidas pela camada *View*.

6.2.3 Camada *View*

Na camada de visualização, *View*, temos uma página que renderiza um mapa com o auxílio do *Google Maps*, e desenha os pontos obtidos por meio de um JSON gerado pelo `MapController`.

Para cada coordenada, composta por latitude e longitude, criamos um elemento `Marker` (da API do *Google Maps*) e, dentro dele, construímos um elemento `LatLng` com a latitude e longitude. Isso é o suficiente para que tenhamos um ponto desenhado no nosso mapa.

No entanto, queremos que o nosso mapa tenha um zoom tal que sempre exiba todos os pontos desenhados na tela. Para isso, instanciamos um elemento `LatLngBounds` que, internamente, constrói um retângulo contendo todos os `LatLng` que forem passados a ele.

Sendo assim, para cada `Marker` construído (que contém um `LatLng`), inserimos esse mesmo `LatLng` no `LatLngBounds`. Ao término da criação dos

`Markers`, dizemos ao nosso elemento `Map` que seus limites estão definidos por `LatLngBounds`.

Com isso, temos um zoom de tal forma que todos os pontos sempre estejam exibidos na tela. Após a renderização inicial do mapa, temos uma função plot que é chamada a cada 10 segundos para atualizar as informações exibidas pelo mapa.

6.2.4 JavaScripts que compõem a View

Como as APIs do Google são disponibilizadas em JavaScript, inicialmente todas as funções de interação com os seus serviços estavam no JSP do mapa. Entretanto, com o aumento da complexidade do projeto e a utilização de mais serviços, tivemos que separar as diversas funções em arquivos.

Atualmente, temos os seguintes JavaScripts:

application.js Principal conjunto de funções, tem como objetivo inicializar o mapa (fazendo uso da API do *Google Maps*) e atribuir ações aos botões de geração de rotas e de localização de aparelhos.

plot.js As funções presentes neste arquivo cuidam do desenho dos pontos no mapa. Além disso, há uma função que converte um endereço digitado como destino em coordenadas que são legíveis para o mapa usando o *Google Geocoding Service*.

route.js Responsável por exibir uma rota entre dois pontos exibidos no mapa por meio de uma requisição ao *Google Directions Service*.

6.3 O otimizador

O módulo de otimização de rotas do Statim é um grande algoritmo genético. De acordo com a modelagem discutida na seção 4, esse módulo segue a seguinte mecânica exibida na figura 16:

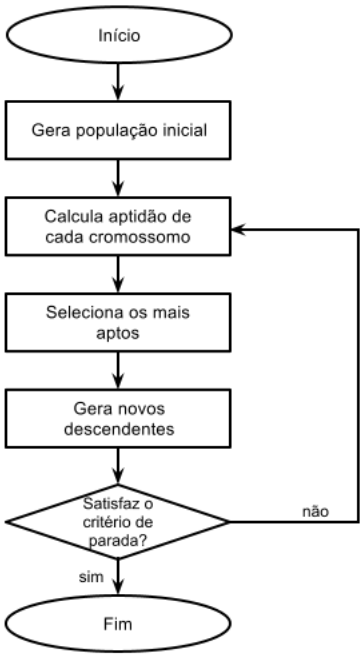


Figura 16: Fluxograma de um algoritmo genético.

6.3.1 Aspectos interessantes da implementação

Como existem muitos operadores interessantes para a seleção dos mais aptos e geração de novos descendentes, precisamos de um código bastante flexível para que possamos trocar esses operadores com facilidade e estudar seus comportamentos. Além disso, não queremos ter que (ou "esquecer de") alterar vários pontos do código para que um operador novo seja incorporado ao otimizador. Esse raciocínio nos levou naturalmente a implementar ambos

operadores como estratégias usando o *Strategy Pattern* (figura 17).

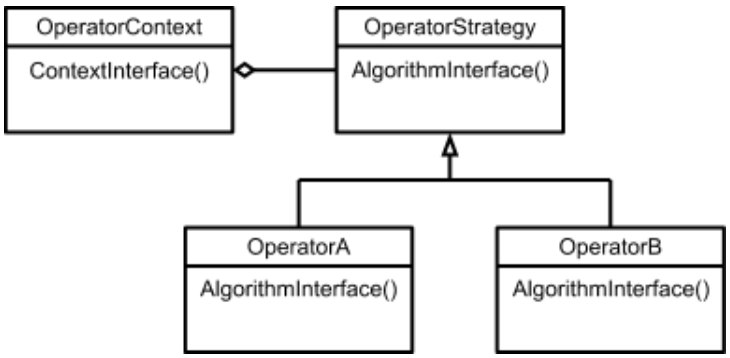


Figura 17: *Strategy Pattern*: Diagrama UML.

Na figura acima, note que é bastante fácil criar novos operadores, pois basta o novo operador implementar a interface `OperatorStrategy`.

Resolvido o problema da flexibilidade do código, há mais um desafio: como serão calculadas as distâncias?

Uma abordagem inicial é enviar várias *requests* ao *Google Directions Service* com um par de coordenadas origem–destino. No entanto, para calcular todos os pares em n coordenadas, faríamos n^2 *requests*! Além disso, tratar essas n^2 respostas num *parser* seria bastante ineficiente.

Após algumas pesquisas, encontramos o *Google Distance Matrix API*, cujo objetivo é justamente diminuir esse número de *requests* para determinar as distâncias entre dois conjuntos de coordenadas (origem e destino). Seu uso é bastante simples: basta enviar os pontos de n pontos de origem e m pontos de destino por meio de uma URL e parsear o JSON (ou XML) de resposta, que representa uma matriz $n \times m$ com as distâncias entre todos os

pares de pontos. Apesar da limitação de 100 pontos por *request*, essa solução se mostrou bem interessante.

Para o *parser* usamos uma biblioteca chamada GSON, que possui, entre outras facilidades, uma forma de se parsear um JSON passando uma classe Java que o representa. Sendo assim, criamos a classe `GoogleDistanceMatrixObject` para representar o JSON de resposta do *Google Distance Matrix Service*.

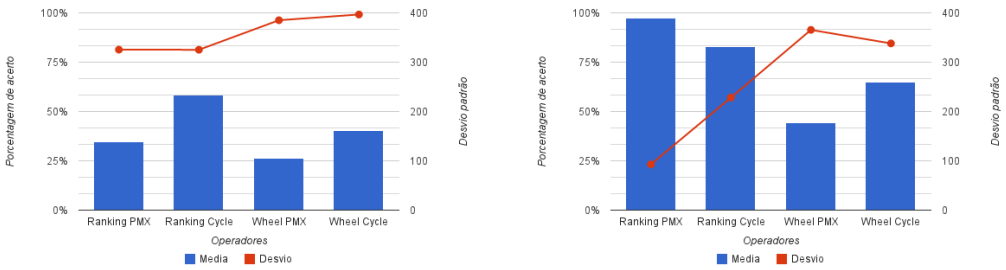
Ao criar esse objeto, temos condições de mapear cada coordenada em inteiros (como discutido na subseção 4.1) para criar os genes, os cromossomos e, além disso, calcular a aptidão para cada cromossomo.

Com o término da implementação de todos os elementos necessários para o funcionamento de um algoritmo genético, iniciamos a criação de um centro de controle, no qual essas partes serão, de fato, usadas para a otimização de rotas. A classe `OptimizerEngine` faz esse papel, dando ao otimizador um funcionamento exatamente como o representado na figura 16.

7 Análise dos resultados

Para analisarmos o desempenho da otimização de rotas do Statim, criamos a classe `BenchmarkEngine` que simula o comportamento do `OptmizerEngine`. Dessa forma, podemos definir o tipo de operador de *crossover* (c) e de seleção (s) que será usado, além do critério de parada (p) e outros parâmetros como: quantidade de pontos (q), número de gerações (n), tamanho da população inicial (i) e quantidade de indivíduos selecionados para reprodução (f).

Como testamos vários cenários diferentes, discutiremos aqueles que achamos mais representativos.



(a) $i = 50$, $n = 50$, $f = 25$ (b) $i = 100$, $n = 50$, $f = 50$

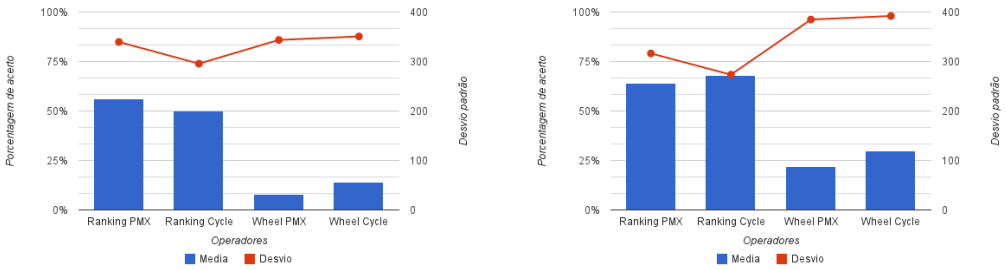
Figura 18: 10000 execuções, $p = generation$, $q = 5$

Nesta primeira comparação, note que apenas um aumento do tamanho da população (i), mantendo a proporção de seleção (f/i) em 50%, foi suficiente para termos uma taxa de acerto de 97,48% no algoritmo de melhor desempenho. Perceba, também, que a seleção dos mais aptos usando o *ranking* apresentou os melhores resultados, mesmo com o perigo da rápida convergência desse método de seleção.

O cenário 18b foi refeito com $n = 100$, apresentando os mesmos resul-

tados. Isso mostra que, nesta configuração, o problema converge para uma solução ótima para algum $n \leq 50$.

Alterando a quantidade de pontos para $q = 10$, tivemos resultados bem interessantes:



(a) $i = 600$, $n = 500$, $f = 300$ (b) $i = 1000$, $n = 500$, $f = 500$

Figura 19: 5000 execuções, $p = generation$, $q = 10$

Note que, em 19a, a combinação *Ranking–PMX* obteve resultados abaixo daqueles obtidos no primeiro caso de teste. Dessa vez, a característica de rápida convergência o levou aos ótimos locais, prejudicando seu desempenho.

No entanto, observe a dupla *Ranking–Cycle*. Apesar da rápida convergência do *ranking*, o *Cycle crossover* permitiu uma maior variabilidade na população, por causar mais modificações nos cromossomos-filhos (veja a seção 4.3.1). Dessa forma, a seleção dos mais aptos foi beneficiada pela variabilidade dos cromossomos, permitindo uma maior exploração do espaço de soluções e, consequentemente, um melhor desempenho em 19b (evidenciado por um menor desvio padrão).

O fraco desempenho do *Wheel* em 19a e 19b pode ser justificado pela sua característica não-elitista de seleção, fazendo com que a solução ótima global seja perdida entre as gerações, mesmo tendo sido encontrada.

8 Parte Subjetiva

8.1 Felipe

8.1.1 Desafios e frustrações

Fazer o TCC numa área que nenhum dos integrantes da dupla tinha grandes conhecimentos foi, sem dúvida, um desafio e um risco. No entanto, com um grande trabalho de pesquisa e motivação pelos resultados, que aos poucos iam aparecendo, conseguimos aprender muito sobre algoritmos genéticos e desenvolver o otimizador sem muitos imprevistos.

O mesmo não posso dizer do módulo *mobile* e *server* do Statim. Como usamos muitos serviços do Google (e, ao longo do ano, ele resolveu alterar alguns), tivemos que parar o desenvolvimento por alguns dias para adequar o projeto às novas políticas. Já, os desafios do *server* estavam todos relacionados à hospedagem: utilizamos muito tempo para pesquisar servidores de hospedagem e, ao decidirmos pelo Heroku, fazer nossa aplicação rodar lá.

8.1.2 Disciplinas cursadas mais relevantes

Destaco abaixo as disciplinas que foram úteis, não só para a construção do Statim, mas para mim, como pessoa e programador:

- **MAC0110 – Introdução à Computação** Apesar da importância mais do que óbvia dessa matéria para todos os programadores, seu destaque é por outro motivo: com o incentivo do professor, participei pela primeira vez de uma maratona de programação. Mesmo estando no primeiro semestre do curso, montamos um time, fomos à competição e tivemos um ótimo desempenho.

- **MAC0122 – Princípios de Desenvolvimento de Algoritmos e MAC0323 – Estruturas de Dados** Essas duas matérias acompanham todo programador, não importa a linguagem ou o projeto. Deixá-las de fora dessa lista seria uma injustiça.

- **MAT0139 – Álgebra Linear para Computação** Essa matéria me marcou não tanto pelo conteúdo (cuja utilidade é indiscutível), mas pelo docente. Obrigado por me ensinar, da pior forma possível, que não devemos depender de professores para aprender algo. Mesmo havendo muitos professores espetaculares no IME, esse ensinamento me acompanhou em muitas outras matérias.

- **MAC0316 – Conceitos Fundamentais de Linguagens de Programação** Nessa matéria, aprendi muitas coisas sobre recursividade e como usá-la correta e eficientemente. Sou um melhor programador depois disso.

- **MAC0342 – Laboratório de Programação Extrema** Uma das poucas matérias na qual podemos programar um projeto grande ~~e tem comida~~. Foi muito interessante a oportunidade de ser *coach* do time que eu fazia parte e contar com a ajuda deles para desenvolver as tarefas propostas. No final, todos fomos *coaches* e aprendemos muito com isso.

- **PCS2590 – Criação e Administração de Empresas de Computação** Com um estilo bastante dinâmico e um professor que, indubitavelmente, conhecia muito da área, essa disciplina me ensinou muito sobre visão de mercado e análise sobre a viabilidade de um projeto de *software*.

8.1.3 Futuro

O módulo de otimização de rotas do Statim foi desenvolvido focando na facilidade em se criar novos operadores e colocá-los em prática para analisar seus resultados. Então, uma ótima ideia seria a implementação de outros operadores de crossover e seleção e analisar seus comportamentos no contexto desse problema.

Além disso, alterar a função de aptidão para incluir o tempo da viagem por uma determinada rota calculada pode, talvez, aprimorar a capacidade de otimização de rotas. Novamente, uma análise seria necessária.

8.1.4 Considerações finais

Ver um projeto que começou numa conversa descompromissada de ônibus tomar forma ao longo dos meses e, ao término do ano, estar funcionando como o planejado é muito recompensador. No entanto, isso não seria possível sem a presença de algumas pessoas que nos ajudaram a pensar no problema:

Gostaria de agradecer ao professor Edson Fregni por nos pressionar com suas perguntas sobre a viabilidade deste projeto nos fazendo pensar em muitos aspectos que "deixaríamos para lá" ou nem havíamos imaginado. Com isso, foi possível ter uma ideia do TCC com detalhes suficientes para minimizarmos as más surpresas ao longo do ano.

Já, no acompanhamento técnico do Statim, meus agradecimentos ao professor Alfredo Goldman, que comprou a ideia logo de cara (talvez rápido demais. Não deu tempo de terminar primeira frase!) e, ao longo do ano, nos

pressionou, apontou erros e acertos, deu dicas... Enfim, nos orientou muito bem, seja por e-mail ou pessoalmente. Muito obrigado!

Por fim, meus agradecimentos aos amigos que fiz e àqueles professores do IME que, mais do que dar aulas, me ensinaram conceitos que levarei comigo para toda a vida.

8.2 Geraldo

8.2.1 Desafios e frustrações

O assunto dessa monografia surgiu meio que por acidente; nós sabíamos o que era Android e algoritmos genéticos pareciam legais para brincarmos, não sabiamos mais nada além disso. Então evidentemente que esse trabalho foi um desafio, mas, como nada que valha a pena vem fácil, tratamos logo de estudar e fomos montando um sistema que foi crescendo. Tomamos muito cuidado ao montá-lo para que fosse de fácil manutenção e com pouco acoplamento.

Ao final tínhamos um belo projeto em Java que não fazia nada, até que numa tarde implementamos a parte genética com tudo que tinhamos feito ate então. Demos o play e funcionou! Aí todo aquele planejamento que fizemos valeu a pena.

Resumindo: o maior desafio foi aprender novas tecnologias e as frustações decorreram de algumas mudanças que tivemos que fazer durante a projeto, mas nada muito significante.

8.2.2 Disciplinas cursadas mais relevantes

Vou falar das disciplinas que usei na construção do Statim e também as que uso com mais frequência. Citarei elas na ordem em que aparecem o curso:

- **MAC0110 – Introdução à Computação, MAC0122 – Princípios de Desenvolvimento de Algoritmos e MAC0323 – Estruturas de Dados** Na minha opinião, essas disciplinas formam a base de um bom programador, não importa a linguagem, tamanho do projeto

ou problema a ser resolvido, constantemente me vejo usando coisas que aprendi nessas matérias.

- **MAC0426 – Sistemas de Bancos de Dados** Em praticamente em todo o projeto que trabalho o banco de dados é uma parte central, seja ele pequeno ou grande, ele sempre está lá. Varias coisas que aprendi nessa matéria uso até hoje, outras uso como base para aprender coisas novas.

- **MAC0332 – Engenharia de Software** Nessa matéria aprendi muito sobre organização e como dar andamento e manutenção a um projeto um pouco maior.

- **PCS2590 – Criação e Administração de Empresas de Computação** Cito essa disciplina por dois motivos: primeiro, foi nessa materia que nosso TCC nasceu; segundo, aprendi a dar um valor (comercial) ao trabalho que faço e pensar de forma diferente nesse aspecto.

8.2.3 Futuro

Acredito que o futuro do projeto esteja na elaboração de mais componentes de cross-over e seleção e na implementação de uma interface mais agradável ao usuário, tanto do servidor quanto do cliente.

8.2.4 Considerações finais

Foi um projeto longo, ao mesmo tempo que foi um projeto que mostrou que o que aprendemos não é por acaso, estamos capacitados a fazer projetos desse nível com uma boa qualidade e finalmente mostrou pra mim, que sempre fui inseguro quanto as minhas capacitações, que estou apto para entrar no mercado de trabalho de forma competitiva.

9 Conclusão

Nesse trabalho estudamos sobre formas de otimizar rotas a partir de algoritmos genéticos, por consequencia estudamos bastante sobre essa classe de algoritmo evolutivo, suas diferentes implementações e eficácia.

Nas técnicas estudadas para implementação do algoritmo genético podemos citar os algoritmos de crossing over, seleção e critério de parada. Com base nos testes efetuados, para algumas combinações desses três operadores encontramos ótimos resultados.

Ao fim, o cálculo da rota ótima fazendo uso de algoritmos genéticos se mostrou muito viável, com taxa de acerto em torno de 90%. Para o prosseguimento desse projeto resta fazermos um refinamento no algoritmo (visando melhorar o desempenho para uma maior quantidade de pontos), além de uma interface mais agradável ao usuário.

A *Google Cloud Messaging*

Em 26 de junho de 2012, foi anunciado o término do suporte ao C2DM[6] para dar lugar ao novo seviço de push notification do Google chamado *Google Cloud Messaging for Android (GCM)*[7].

O GCM é muito parecido com o seu antecessor, com a vantagem de que agora o envio de mensagens a devices é muito mais fácil, pois esse novo serviço disponibiliza uma API própria de alto nível para facilitar o trabalho que, com o C2DM, era implementado manualmente. Apesar disso, as aplicações que usam o C2DM podem ser migradas facilmente para o GCM sem a necessidade do uso dessa nova API.

Uma das prinicipais características do GCM é a ausência de cotas de mensagens enviadas, ou seja, podemos enviar quantas mensagens quisermos - ao contrário do C2DM, que tinha uma limitação, ainda que alta, de 200.000 mensagens por dia. Agora, também, podemos escolher enviar mensagens em formato JSON ou plain text, basta passar no header do POST ao GCM o seguinte Content-Type:

`Content-Type: application/json` (para JSON)

ou

`Content-Type: application/x-www-form-urlencoded;charset=UTF-8` (para *plain-text*)

Com essa novidade do GCM, as mensagens que trafegam pelo Statim (coordenadas e itinerários) são no formato JSON, que permite uma maior robustez e facilidade de *parsing* quando comparado ao *plain-text*.

B Timeline

22/05/12: Início do projeto.

29/05/12: Término das principais funcionalidades de rastreamento.

01/06/12: Início dos estudos sobre hospedagem no Heroku, OpenShift e Amazon.

04/06/12: Adaptação do Statim *Server* para hospedagem no Heroku.

06/06/12: Statim *Server* hospedado no Heroku.

07/06/12: Início da monografia.

25/06/15: Término da versão preliminar da monografia.

27/06/12: Google descontinuou o C2DM para dar origem ao *Google Cloud Messaging (GCM)*.

10/07/12: Migração do Statim *Mobile* para o GCM.

16/07/12: Inclusão do itinerário nos *devices*.

19/07/12 - 24/09/12: Modelagem matemática para otimização de rotas.

25/09/12 - 01/10/12: Modelagem genética para o otimizador.

02/10/12 - 08/10/12: Implementação do otimizador.

11/10/12 - 25/10/12: Compilação e análise dos dados gerados pelo otimizador.

22/10/12 - 26/10/12: Criação do pôster.

30/10/12 - 11/11/12: Criação dos slides para apresentação do trabalho.

13/11/12: Apresentação do trabalho desenvolvido.

28/11/12: Finalização e revisão da monografia.

Referências

[1] RUDIN, Andy. Where's my Gphone? **Google Official Blog**, November 5, 2007. Disponível em: <http://googleblog.blogspot.com.br/2007/11/wheres-my-gphone.html>. Acesso em: 25/06/2012.

[2] ANDROID, the world's most popular mobile platform. **Android Developers**. Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>. Acessado em: 25/06/2012.

[3] OCHI, Luiz Satoru. Algoritmos Genéticos: origem e evolução. **Sociedade Brasileira de Matemática Aplicada e Computacional**. 19 de Setembro de 2006. Disponível em: <http://sbmac.org.br/bol/bol-2/artigos/satoru/satoru.html>. Acessado em: 20/05/2012.

[4] GORNI, Henrique. Algoritmos Genéticos - Visão Geral. **Demiurgo**, 20 de Maio de 2009. Disponível em: <http://www.demiurgo.com.br/2009/05/20/algoritmos-geneticos-visao-geral/>. Acessado em: 24/05/2012.

[5] OBITKO, Marek. Genetic Algorithms. **Introduction to Genetic Algorithms**, 1998. Disponível em: <http://www.obitko.com/tutorials/genetic-algorithms/>. Acessado em: 24/05/2012

[6] ANDROID, cloud to device messaging framework. **Google Developers**. 3 de Julho de 2012 Disponível em: <https://developers.google.com/android/c2dm/>. Acessado em: 23/09/2012.

[7] GOOGLE, Cloud Messaging for Android. **Android Developers**. 13 de Setembro de 2012 Disponível em: <http://developer.android.com/guide/google/gcm/index.html>. Acessado em: 23/09/2012.

[8] T. Starkweather and S. Mcdaniel and D. Whitley and K. Mathias and C. Whitley. *A Comparison of Genetic Sequencing Operators.* Proc of the 4th Int. Conf. Genetic Algorithms. p. 69–76. 1991.

[9] I. Oliver, D. Smith, and J. R. Holland. *A study of permutation crossover operators on the traveling salesman problem.* Genetic Algorithms and their Applications: Proc. of the 2nd Int. Conf, John Grefenstette. p. 224–230. 1987.

[10] ÜÇOLUK, Göktürk. *Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation.* Intelligent Automation and Soft Computing, 3(8), TSI Press. p. 265–272. 2002.