Dissertação de Mestrado

Algoritmos para Atribuição de Tráfego em Redes de Filas Finitas Dependentes do Estado

por

Renato de Almeida Nascimento

Setembro de 2011

Renato de Almeida Nascimento

Algoritmos para Atribuição de Tráfego em Redes de Filas Finitas Dependentes do Estado

Dissertação apresentada ao Programa de Pós-Graduação em Estatística do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial à obtenção do título de Mestre em Estatística.

Orientador: Frederico R. B. Cruz

Universidade Federal de Minas Gerais
Belo Horizonte, setembro de 2011

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Estatística
Programa de Pós-Graduação
Caixa Postal 702
31270-901 Belo Horizonte- MG – Brasil

Telefone (31) 3409-5923
Fax (31) 3409-5924
E-mail: pgest@ufmg.br
WEB: http://www.est.ufmg.br/posgrad/

FOLHA DE APROVAÇÃO

Algoritmos para Atribuição de Tráfego em Redes de Filas Finitas Dependentes do Estado

Renato de Almeida Nascimento

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. FREDERICO RODRIGUES BORGES DA CRUZ
(Orientador/Departamento de Estatística/UFMG);

PROF. ANDERSON RIBEIRO DUARTE
(Departamento de Matemática/UFOP);

PROFª. PAULA DE CAMPOS OLIVEIRA
(FACISA BH).

Belo Horizonte, 16 de Setembro de 2011.

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Estatística
Programa de Pós-Graduação
Caixa Postal 702
31270-901 Belo Horizonte- MG – Brasil

Telefone (31) 3409-5923
Fax (31) 3409-5924
E-mail: pgest@ufmg.br
WEB: http://www.est.ufmg.br/posgrad/

ATA DA DEFESA DE DISSERTAÇÃO DO ALUNO
Renato de Almeida Nascimento

Realizou-se, no dia 16 de Setembro de Dois mil e Onze, às 14:00 horas, na Sala 2025 do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais, a 166ª defesa de dissertação de Mestrado em Estatística, intitulada **"Algoritmos para Atribuição de Tráfego em Redes de Filas Finitas Dependentes do Estado"**, apresentada por **Renato de Almeida Nascimento**, como requisito parcial para a obtenção do grau de Mestre em Estatística, à seguinte Comissão Examinadora: PROF. FREDERICO RODRIGUES BORGES DA CRUZ – ORIENTADOR (DEPARTAMENTO DE ESTATÍSTICA – UFMG); PROF. ANDERSON RIBEIRO DUARTE –(DEPARTAMENTO DE MATEMÁTICA – UFOP); PROFª. PAULA DE CAMPOS OLIVEIRA(FACISA BH).

A Comissão considerou a dissertação:

(X) Aprovada
( ) Aprovada condicionalmente, sujeita a alterações, conforme folha de modificações, anexa
( ) Reprovada, conforme folha de modificações, anexa

Finalizados os trabalhos, lavrei a presente ata que, lida e aprovada, vai assinada por mim e pelos membros da Comissão examidora:
Belo Horizonte, 16 de Setembro de 2011.

Prof. Frederico Rodrigues Borges da Cruz
(Orientador/Departamento de Estatística/UFMG);

Prof. Anderson Ribeiro Duarte
(Departamento de Matemática/UFOP);

Profª. Paula de Campos Oliveira
(FACISA BH).

Resumo

Os modelos de tráfego precisam ser detalhados o suficiente para capturar as sutilezas fundamentais envolvidas. Precisam também ser simples o bastante para se enquadrarem bem em um arcabouço de otimização. Nesta dissertação investigamos um modelo de atribuição de tráfego de veículos, o clássico modelo do ótimo do sistema (do inglês, *system optimum*), baseado em filas finitas configuradas em redes. O modelo de filas utilizado representa satisfatoriamente situações de congestionamento, isto é, situações em que a velocidade de um usuário decai com o aumento do número de usuários simultâneos no sistema. Obtivemos resultados com uma heurística do tipo evolucionária que indicaram que as atribuições de tráfego produzidas são coerentes e robustas.

Palavras-chave: Ótimo do sistema, redes de filas, filas finitas, sistemas estocásticos.

Abstract

The traffic models must be detailed enough to capture the fundamental details involved. Also these models must be simple enough to fit well into an optimization framework. In this dissertation we investigate a traffic assignment model, the classical *System Optimum* (SO) model, which is based on finite queueing networks. The queueing model chosen represents quite well those situations with congestion effects in which the speed of an user decays with the increase of the number of users simultaneously in the system. We obtained results with and evolutionary heuristics that indicated that the traffic assignments make sense and are robust.

Keywords: System optimum, queueing networks, finite queues, stochastic systems.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação e Definição do Problema

Pesquisadores da área vêm fazendo tentativas sucessivas para modelar o mais precisamente possível a seleção de rotas, por parte dos usuários, em redes congestionadas de tráfego de veículos (veja, por exemplo, o trabalho de Helbing et al., 2005). Comumente duas classes de modelos são consideradas, o modelo do equilíbrio do usuário (UE, do inglês *User Equilibrium Model*) e o modelo do ótimo do sistema (SO, do inglês *System Optimum Model*).

O modelo EU assume o perfeito conhecimento dos tempos de viagem e que cada um dos usuários irá escolher aquela rota que minimiza apenas seu próprio tempo de percurso. Por outro lado, o modelo SO assume que todos os usuários são capazes de cooperar entre si (ou são 'induzidos' a cooperar, via direcionamento do tráfego), a fim de minimizar o tempo total de viagem no sistema como um todo.

O foco desta dissertação é o modelo SO, em que, basicamente, dada uma rede de tráfego como o exemplo apresentado na Figura 1.1, o que precisa ser feito é a determinação das probabilidades de roteamento p_{a_1} e p_{a_2}, para ir de A até B, de tal forma a minimizar os custos globais do deslocamento.

O modelo SO pode ser formulado como se segue (detalhes podem ser

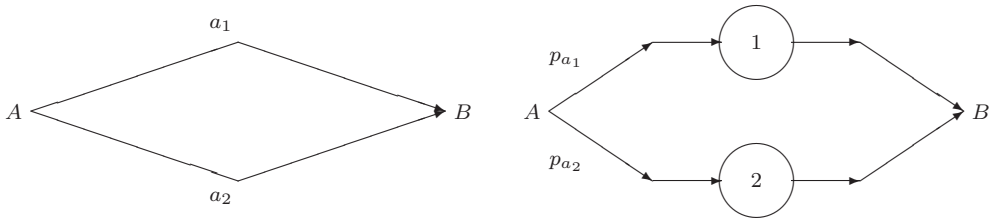

Figura 1.1: Uma rede com dois trechos e o respectivo modelo $M/G/c/c$

encontrados, por exemplo, em Sheffi, 1985):

$$\min \sum_{\forall a} x_a c_a(x_a), \tag{1.1}$$

sujeito a:

$$\begin{aligned} \sum_k f_k^{rs} &= q^{rs}, \ \forall\, r, s, & (1.2)\\ x_a &= \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs}, \ \forall\, a, & (1.3)\\ f_k^{rs} &\geq 0, \ \forall\, k, r, s, & (1.4) \end{aligned}$$

em que x_a é o fluxo no trecho a, $c_a(x_a)$ é o tempo de percurso no trecho a, em função do fluxo x_a neste percurso, f_k^{rs} é o fluxo na rota k entre a origem r e o destino s, q^{rs} é a demanda entre r e s e $\delta_{a,k}^{rs}$ é uma variável indicadora que assume o valor 1, se o arco a está no caminho k entre r e s, ou o valor 0, caso contrário (maiores detalhes sobre os modelos SO e UE, bem como suas variantes e combinações, podem ser encontrados em Sheffi, 1985).

Um grande problema, em qualquer destas duas classes de modelos, é que o tempo de percurso, a principal medida de qualidade, é usualmente considerado determinístico ou senão proveniente de aproximações de modelos estocásticos. Os mais importantes modelos de tempo de percurso, construídos nos últimos 40 anos, podem ser observados na Figura 1.2, que apresenta o

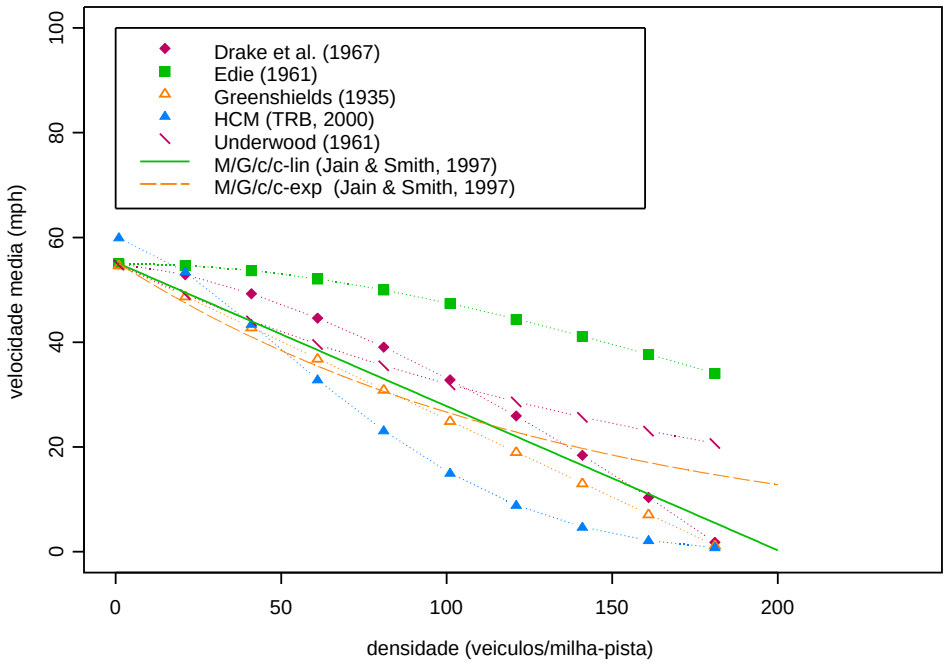

Figura 1.2: Tempos de serviço empíricos para tráfego de veículos (Drake et al., 1967; Edie, 1961; Greenshields, 1935; TRB, 1985; Underwood, 1961) e modelos $M/G/c/c$ dependentes do estado (Jain & Smith, 1997)

resultado de vários estudos empíricos em rodovias norte americanas.

Redes de filas finitas aplicadas aos modelos SO e UE apenas começam a ser estudas. Muitas das questões envolvidas são ainda tópico de pesquisa em aberto. Em particular, estamos interessados em estudar aqui o modelo de atribuição de tráfego SO no qual os tempos de percurso sejam modelados por redes de filas $M/G/c/c$ dependentes do estado (ver Figura 1.2). Na notação de Kendall, M denota um processo markoviano de chegada, G, um tempo de serviço com distribuição geral e dependente do estado e c representa tanto o número de servidores em paralelo quanto o espaço total na fila, incluindo os usuários que estão em serviço (ver Jain & Smith, 1997).

1.2 Objetivos e Escopo da Dissertação

Um dos objetivos desta dissertação é contribuir com o desenvolvimento de algoritmos para a melhoria do desempenho de sistemas de interesse prático, através de uma modelagem matemática sofisticada que considera a estocasticidade natural de tais sistemas. Em particular, nosso objetivo principal nesta dissertação é estudar algoritmos para atribuição de tráfego em redes de transporte modeladas por filas finitas com taxa de serviço dependentes do estado (isto é, a distribuição do tempo de serviço depende do número total de usuários presentes no sistema e seu valor esperado decai com o aumento do número de usuários simultâneos).

Uma abordagem preferencialmente computacional é adotada para a estimação das medidas de desempenho das redes de filas finitas consideradas (para detalhes, ver Artalejo & Gómez-Corral, 2008). Métodos de programação matemática fundamentam o algoritmo de otimização selecionado (Hillier & Lieberman, 2005), que é do tipo evolucionário (Goldberg, 1989).

1.3 Contribuições

A metodologia que ajudamos a desenvolver, descrita nessa dissertação, servirá de suporte para estudos futuros envolvendo sistemas estocásticos nos quais estejam presentes taxas de serviço que sejam dependentes do estado, em particular aqueles sistemas em que esta taxa decai com o aumento do número de usuários simultâneos. Em um arcabouço multi-objetivo, a metodologia desenvolvida deverá atuar como uma importante ferramenta auxiliar que possibilitará a expansão do leque de aplicações.

Destacamos que a linguagem de programação utilizada foi o C++, reconhecidamente um ambiente poderoso e moderno, que permite, entre outras

vantagens, a inclusão de novas funcionalidades. Dessa forma, podemos esperar que a teoria aqui detalhada constitua-se um importante alicerce para futuros estudos que visem a melhoria da qualidade de serviços, que é tão importante no competitivo mundo globalizado.

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma. No Capítulo 2 apresentamos uma revisão da literatura e os algoritmos utilizados para a análise de desempenho em redes de tráfego e sua otimização. No Capítulo 3 apresentamos alguns resultados de experimentos computacionais realizados para validação do algoritmo de otimização utilizado. Finalmente, no Capítulo 4 são apresentadas as principais conclusões alcançadas, bem como observações finais.

Capítulo 2

Revisão da Literatura

2.1 Introdução

O modelo SO (e similares) não é recente. Já foi estudado anteriormente por vários pesquisadores. Diversos algoritmos foram propostos para sua resolução, tanto algoritmos do tipo exato quanto do tipo heurísticos (veja Cruz et al., 2010, para um revisão).

Para resolução do modelo SO, serão utilizados aqui algoritmos heurísticos do tipo evolucionários. Mais especificamente, usaremos um algoritmo do tipo *Differential Evolution* (DE), membro da família dos algoritmos genéticos. Algumas das propriedades do DE que o tornam adequado ao modelo SO são a robustez, a rapidez, a facilidade de uso e a habilidade em operar bem em superfícies planas (Price & Storn, 2010).

Dado que o algoritmo DE fornece a cada iteração um vetor de roteamento candidato $\mathbf{p} = (\ldots, p_a, \ldots)$, medidas de desempenho precisam ser determinadas, para o cálculo do respectivo valor da função objetivo, o que será feito via modelos de filas finitas. A descrição do método de obtenção das medidas de desempenho é apresentada na Seção 2.2. Os passos do algoritmo DE serão detalhados na Seção 2.3.

2.2 Medidas de Desempenho

Para um trecho *único* de via, podemos considerar que há c servidores em paralelo, que é também a capacidade total de tal trecho. Esse número máximo de usuários (veículos, pedestres, ...) permitidos no sistema pode ser obtido da expressão:

$$c = \lfloor \kappa l w \rfloor, \tag{2.1}$$

em que:

κ := é a densidade máxima de congestionamento (do inglês *jam density parameter*, definido em veículos/milha-pista ou veículos/km-pista), normalmente situando-se na faixa de 185 a 265 veículos/milha-pista;

l := distância do segmento de via (em milhas ou km);

w := largura do segmento de via (em número de pistas);

$\lfloor x \rfloor$:= é a função piso, isto é, o maior inteiro não superior a x.

Lembramos que os modelos mais usuais de tempo de percurso não levam em consideração as dimensões (distância e largura, que são finitas) do trecho de via (veja, por exemplo, Bureau of Public Roads, 1964; Akçelik, 1991).

Assim, podemos dizer que, em geral, a taxa de serviço μ é função da velocidade média V_i na qual todos os i indivíduos, presentemente no sistema naquele momento, percorrem o trecho da via. Toma, portanto, um tempo médio de $t_i = l/V_i$ (segundos, horas, ...), para cada um dos i usuários percorrer o segmento, em que i, novamente, é o número de usuários presentes simultaneamente neste segmento.

Para a dedução de um modelo exponencial para a taxa de serviço (também seria possível utilizar um modelo linear, fazendo as mudanças necessárias,

conforme pode ser visto na Figura 1.2, pág. 3), assumimos que a taxa de serviço de cada um dos n servidores ocupados, $r_n = V_n/l$, é relacionada com o número de usuários por meio de uma função exponencial. A forma desta função exponencial é baseada na relação usuário-velocidade, conforme apresentado na Eq. (2.2),

$$V_n = V_1 \exp\left[-(\frac{n-1}{\beta})^{\gamma}\right]. \tag{2.2}$$

Os parâmetros β e γ são denominados parâmetros de escala e de forma, respectivamente, e podem ser encontrados pelo ajuste de alguns dos pontos das curvas apresentadas na Figura 1.2. Pela aproximação cuidadosa das posições de três pontos representativos das curvas da Figura 1.2, $(1, V_1)$, (a, V_a) e (b, V_b), temos as relações algébricas abaixo (para maiores detalhes, veja Jain & Smith, 1997):

$$\gamma = \ln\left[\frac{\ln(V_a/V_1)}{\ln(V_b/V_1)}\right] / \ln(\frac{a-1}{b-1}), \tag{2.3}$$

$$\beta = \frac{a-1}{[\ln(\frac{V_1}{V_a})]^{1/\gamma}} = \frac{b-1}{[\ln(\frac{V_1}{V_b})]^{1/\gamma}}. \tag{2.4}$$

Podemos selecionar diferentes pontos, para modelar diferentes situações, com diferentes velocidades máximas V_1 (isto é, a velocidade de um ocupante único no sistema).

Combinando as Equações (2.2), (2.3) e (2.4), temos:

$$r_n = \frac{V_1}{l} \exp\big[-(\frac{n-1}{\beta})^{\gamma}\big], \tag{2.5}$$

em que V_1 é a velocidade para um ocupante sozinho no sistema.

A taxa de serviço total do sistema é equivalente à taxa de serviço de cada servidor multiplicada pelo número de servidores em operação (isto é, o

número de servidores que estão ocupados). Podemos, portanto, expressar a taxa de serviço global por:

$$\mu_n = nr_n = n\frac{V_1}{l}\exp\left[-\left(\frac{n-1}{\beta}\right)^{\gamma}\right]. \tag{2.6}$$

Admitindo-se um modelo de filas $M/G/c/c$ dependente do estado, teremos que a expressão para a distribuição de probabilidades do número de usuários no sistema pode ser obtida pela substituição da Eq. (2.6) nas equações de Chapman-Kolmogorov, para uma única fila:

$$p(n) = \frac{\lambda^n}{\Pi_{i=1}^{n}\mu_i}p(0), \tag{2.7}$$

para $n = 1, \ldots, c$, em que λ é taxa de chegada e $p(0)$ é a probabilidade de o sistema estar vazio, dada por

$$p(0)^{-1} = 1 + \sum_{n=1}^{C}\left\{\frac{\lambda^n}{\left[\Pi_{i=1}^{n}\mu_i\right]}\right\}. \tag{2.8}$$

Assim, substituindo-se a Eq. (2.6) nas Equações (2.7) e (2.8), tem-se:

$$p(n) = \frac{\lambda^n}{\Pi_{i=1}^{n} i\left(\frac{V_1}{L}\right)\exp\left\{\left[-\left(\frac{i-1}{\beta}\right)^{\gamma}\right]\right\}}p(0), \tag{2.9}$$

em que

$$p(0)^{-1} = 1 + \sum_{n=1}^{C}\left\{\frac{\lambda^n}{\Pi_{i=1}^{n} i\left(\frac{V_1}{L}\right)\exp\left[-\left(\frac{i-1}{\beta}\right)^{\gamma}\right]}\right\}. \tag{2.10}$$

Note que V_1 pode ser expresso em quilômetros por hora (km/h) ou em milhas por hora (mph), l pode ser expresso em quilômetros ou em milhas e λ é expresso em horas^{-1}.

Conhecendo a distribuição de probabilidades do número de usuários no sistema, podemos calcular a taxa de chegada efetiva, dada por:

$$\theta = \lambda\big[1 - p(c)\big], \tag{2.11}$$

em que $p(c)$ é a probabilidade de bloqueio, isto é, $p(c) \equiv \Pr[N = c]$.

O número esperado de usuários no sistema L pode ser determinado diretamente da definição de esperança matemática de uma variável aletória, isto é,

$$L = \sum_{n=0}^{c} np(n). \tag{2.12}$$

Finalmente, podemos determinar o atraso esperado W pela Lei de Little:

$$W = \frac{L}{\lambda}, \tag{2.13}$$

mas como existe uma capacidade finita, que produz o fenômeno do bloqueio (isto é, alguns usuários podem se rejeitados se chegarem no sistema quando ele estiver com a capacidade c esgotada), temos que:

$$W = \frac{L}{\theta}. \tag{2.14}$$

Concluindo, para o modelo exponencial (ou linear), como $t \equiv W$, temos:

$$t = \theta^{-1} \sum_{n=0}^{c} np(n). \tag{2.15}$$

Notamos que uma vez conhecido o vetor de probabilidades de roteamento $\mathbf{p} = (\ldots, p_a, \ldots)$, não é difícil determinar o valor da função objetivo, dada pela Eq. (1.1), $z(\mathbf{x}) \equiv \sum_{\forall a} x_a c_a(x_a)$. De fato, como $x_a \equiv \theta$ e $c_a(x_a) \equiv t$, calculamos $z(\mathbf{x})$ facilmente por meio das Equações (2.11) e (2.15).

Ressaltamos, entretanto, que o problema de determinação de medidas de desempenho está resolvido apenas parcialmente. A obtenção de medidas para *redes* de filas $M/G/c/c$ é uma tarefa consideravelmente mais complexa. Detalhes não serão dados aqui. Apenas diremos que será adotada uma abordagem computacional aproximada para obtenção destas medidas. Mais especificamente, utilizaremos o método da expansão generalizado, que é uma combinação de métodos de tentativa-e-erro e decomposição nó-a-nó (para detalhes, veja Jain & Smith, 1997).

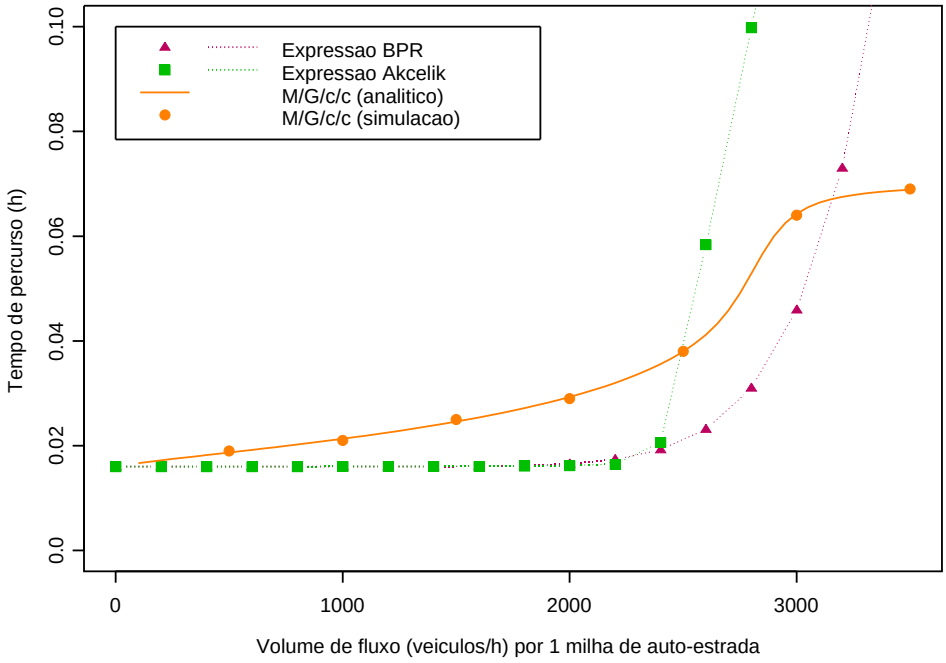

Figura 2.1: Fluxos em um trecho de uma milha (Cruz et al., 2010)

Um último fato é importante mencionarmos. Se por um lado a utilização do modelo $M/G/c/c$ dependente do estado é mais aderente à realidade, por outro este modelo complica bastante o problema de otimização. De fato,

conforme vemos na Figura 2.1, uma curva clássica (do tipo BPR) de tempo de percurso em função do volume de tráfego difere bastante daquela produzida pelo modelo $M/G/c/c$ dependente do estado, especialmente sob tráfego pesado. Sob tráfego leve, a abordagem via filas finitas é bastante próxima da fórmula BPR clássica, mas o modelo $M/G/c/c$ dependente do estado produz curvas de tempos de percurso com formato S, que representa problemas sérios para qualquer algoritmo de otimização, pela não-convexidade. Este formato em S é consequência direta da capacidade (considerada limitada, pelo modelo de $M/G/c/c$ dependente do estado) do trecho de via considerado.

2.3 Heurística de Otimização

Conforme já mencionado, para determinação das probabilidades de roteamento ótimas, $\mathbf{p}^* = (\ldots, p_a^*, \ldots)$, empregaremos uma heurística do tipo DE, que faz parte da ampla família dos algoritmos evolucionários. As seguintes características do algoritmo DE, válidas para problemas de otimização em espaço contínuo (Storn & Price, 1997), justificam sua utilização como método de solução adequado ao modelo SO:

- é simples, rápido e robusto;
- possui a capacidade de encontrar ótimos globais;
- pode ser facilmente implementado em ambiente computacional paralelo, o que acelera a otimização;
- é eficaz em problemas de otimização não linear e pode ser facilmente adaptado para problemas de otimização mistos;
- não requer função objetivo diferenciável;

- opera em superfícies suaves;
- consegue gerar múltiplas soluções em uma única execução.

Além disso, Babu & Sastry (1999) demonstraram que a técnica DE foi o melhor algoritmo do tipo evolucionário, após o estudo de sete problemas bem difíceis de programação matemática inteira mista, na área de planejamento e controle em engenharia química. Lampinen & Zelinka (1999) reportaram, para um conjunto de difíceis funções-objetivo não-lineares com múltiplas restrições não-triviais, que as soluções obtidas pelo DE superaram qualquer outro método experimentado (*branch-and-bound* com programação quadrática sequencial, programação não-linear contínua discreta inteira, *simulated annealing*, algoritmos genéticos, programação não-linear inteira mista, ...).

Os algoritmos DE são uma versão de algoritmo genético, da classe dos algoritmos evolucionários, baseados no princípio da sobrevivência do mais apto. São basicamente algoritmos de otimização e busca baseados em populações. O DE difere de um algoritmo evolucionário na forma de uma mutação diferencial. Em um algoritmo evolucionário, a mutação é baseada na saída de uma função de distribuição pré-definida, enquanto o DE usa a diferença de vetores-objeto aleatoriamente amostrados. Ao invés de usar apenas informação local de cada vetor-objeto (indivíduos da população), o DE muta todos os vetores-objeto com a mesma probabilidade. Deste modo o espaço de busca é coberto mais completamente na tentativa de encontrar um ótimo global.

O método é definido como um método de busca direta e paralela que opera em uma população P_G de tamanho constante que é associada com cada geração G e consiste de NP vetores, ou soluções candidatas, $\mathbf{X}_{p,G}$, $p = 1, 2, \ldots, NP$. Cada vetor $\mathbf{X}_{p,G}$ consiste de D variáveis de decisão $X_{o,p,G}$, $o = 1, 2, \ldots, D$.

O acima exposto pode ser brevemente resumido como:

$$\begin{aligned} P_G &= \{\mathbf{X}_{1,G}, \mathbf{X}_{2,G}, \ldots, \mathbf{X}_{p,G}, \ldots, \mathbf{X}_{NP,G}\}, \\ \mathbf{X}_{p,G} &= \{X_{1,p,G}, X_{2,p,G}, \ldots, X_{o,p,G}, \ldots, X_{D,p,G}\}, \\ G &= 1, \ldots, G_{\max}, \\ NP &\geq 4. \end{aligned}$$

Cada probabilidade de roteamento é então considerada como a variável de decisão $p_a \equiv X_{o,p,G}$.

Os diferentes passos do algoritmo são:

Passo 1: Escolha uma estratégia $DE/x/y/z$, de acordo com a notação de Price & Storn (2010), em que x especifica o vetor a sofrer mutação (pode ser *rand*, se o vetor é escolhido aleatoriamente na população, ou *best*, se o vetor é escolhido como o de menor custo na população atual), y é o número de vetores-diferenças usados e z especifica o esquema de cruzamento (pode ser *bin*, se o cruzamento é através de experimentos binomiais independentes, ou *exp*, se o cruzamento é através de experimentos exponenciais). Price & Storn (2010) sugerem 10 estratégias diferentes para o DE (por exemplo, $DE/rand/1/bin$, $DE/rand/2/bin$, $DE/current\text{-}to\text{-}rand/1$, e assim por diante).

Passo 2: Inicialize os parâmetros de controle. Os parâmetros de controle definidos pelo usuário, os quais permanecem constante durante o processo de busca, são a constante de cruzamento CR, o tamanho da população NP, o fator de escala de mutação F e o numero máximo de gerações $G_{\max}$.

Passo 3: Inicialize a população. A população inicial $P_{G=0}$ nos fornece uma solução de partida, para a busca do ótimo, e é escolhida aleatoriamente entre os limites dos parâmetros que são definidos pelas restrições. Deve cobrir todo espaço de busca.

Passo 4: Avalie a adequabilidade de cada vetor e encontre aquele com maior adequabilidade. A função objetivo tem que ser avaliada para cada vetor na população, após o qual o melhor pode facilmente ser determinado.

Passo 5: Realize mutação e cruzamento. A mutação é quando o DE gera novos vetores (vetores mutantes), pela adição da diferença ponderada entre dois ou mais vetores escolhidos aleatoriamente da população atual, P_G, a um terceiro vetor (vetor alvo). A mutação pretende manter uma população robusta e procura uma nova área. O vetor mutante será então usado para construir a população da próxima geração. A mutação gera uma população de vetores de acordo com a equação:

$$\mathbf{V}_{i,G+1} = \mathbf{X}_{r_1,G} + F \times (\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G}) \,,$$

em que $\mathbf{V}_{i,G+1}$ é o vetor mutante, $\mathbf{X}_{i,G}$, para $i = 1, 2, \ldots, NP$, é o vetor alvo, $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$ são os índices aleatórios e $F \in [0, 2]$ é o fator de escala de mutação.

Recombinação, ou cruzamento, é complementar a mutação e constrói vetores tentativas fora dos parâmetros dos vetores objeto existente a fim de reforçar os sucessos anteriores. A operação de cruzamento cria um vetor tentativa $\mathbf{U}_{p,G+1}$ selecionando elementos do vetor alvo $\mathbf{X}_{p,G}$ e do vetor mutante $\mathbf{V}_{p,G+1}$. A constante de cruzamento CR controla a probabilidade de que um parâmetro do vetor tentativa venha do vetor mutante $\mathbf{V}_{p,G+1}$, em vez do vetor atual $\mathbf{X}_{p,G}$, e portanto varia de 0 a 1.

Passo 6: Verifique os limites superiores e inferiores das variáveis. No caso presente, os valores dos vetores tentativa devem ser conferidos para garantir que ficaram entre 0 e 1 (que são as fronteiras das probabilidades de roteamento). Se um parâmetro que sofreu mutação excede alguma das restrições de fronteira, um caminho é selecionar um novo valor aleatório que seja adequado.

Passo 7: Execute a seleção. Para selecionar os vetores da próxima geração, cada vetor tentativa deve ser avaliado e comparado com o vetor alvo, por meio do valor da função objetivo. Se a adequabilidade do vetor tentativa é melhor que a do vetor alvo, o vetor tentativa passa a ser o novo vetor alvo. Caso contrário, o vetor alvo é retido para próxima geração. Como resultado, todos os indivíduos da próxima geração são tão bons quanto (ou melhores que) os indivíduos da geração atual.

Passo 8: Repetir o ciclo evolucionário, até que $G_{\max}$ seja alcançado.

Sugerimos consultar Lampinen & Zelinka (1999), Lampinen (2000) ou Fan & Lampinen (2004), para maiores detalhes sobre o sistema de mutação, os valores para os parâmetros de controle e o critério de parada. Diremos aqui apenas que consideramos $CR = 0.8$, $NP = 8$, $F = 1$ e $G_{\max} = 1.000$. Para a descrição de um estudo mais completo e detalhado sobre a seleção destes valores, recomendamos consultar Cardoso (2010).

Capítulo 3

Resultados Experimentais

3.1 Introdução

Os algoritmos utilizados foram codificados em C++. O programa está disponível no Apêndice A, para ensino e pesquisa. Os experimentos foram conduzidos em um PC, no Windows© Vista. Passemos a seguir para a apresentação dos resultados de uma análise empírica preliminar do desempenho do algoritmo DE.

3.2 Análise do Algoritmo

Realizamos um análise empírica do desempenho do algoritmo, DE em termos do tempo de processamento e do número de variáveis de decisão. Foram experimentadas redes de duas a sete variáveis de decisão, em topologia divisão (veja na Figura 3.1, a rede testada, com sete variáveis de decisão). Os resultados estão sumarizados na Tabela 3.1.

O interesse foi apenas nos tempos de processamento. Para nossa comodidade, foram obtidos resultados para os tempos de processamento para apenas duas execuções por número de variáveis de decisão. O número de replicações é baixo, mas serve aos nossos propósitos de fazer uma análise preliminar do

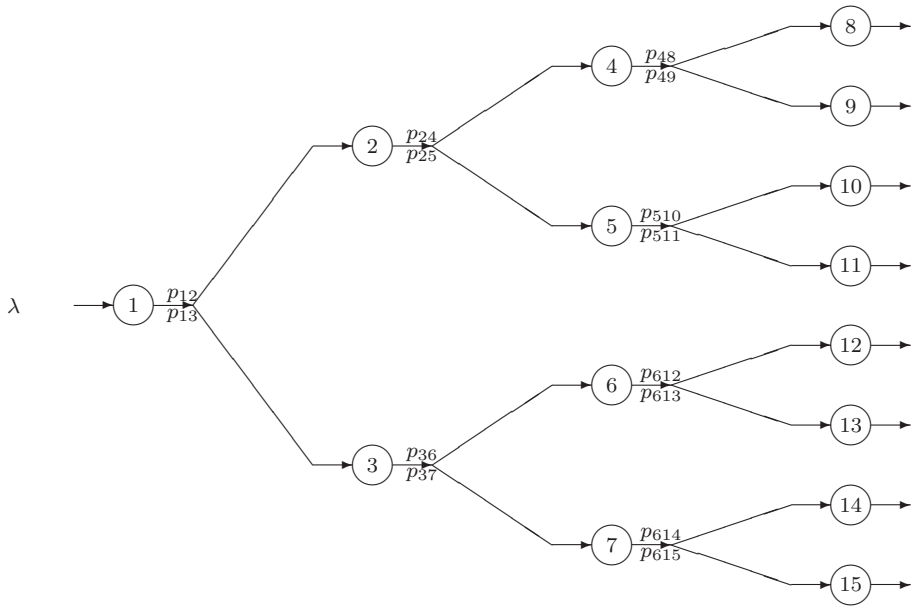

Figura 3.1: Rede de sete variáveis de decisão em topologia divisão

Tabela 3.1: Tempo de processamento em função do número de variáveis

	ucp (s)	
Número de variáveis	Execução 1	Execução 2
1	0,10	0,10
2	2,0	1,0
3	32	32
4	26	25
5	70	69
6	87	86
7	111	110

algoritmo. De fato, podemos notar que esses tempos crescem com o aumento do número de variáveis, mas para os casos testados esse crescimento não foi explosivo. Essa característica mostra a robustez e adequação do DE ao problema SO em questão.

3.3 Resultados de Atribuição de Tráfego

Utilizamos como exemplo a rede da Figura 1.1 (página 2). Os pontos A e B estão conectados por duas rotas alternativas, sendo que uma delas é mais curta (e consequentemente mais rápida) que a outra. A configuração desta rede é apresentada na Tabela 3.2. Resultados de alocações ótimas podem ser vistos na Tabela 3.3, para diferentes taxas de chegada (veja Carvalho et al., 2011).

Tabela 3.2: Configuração para rede de dois trechos

Rota	l^*	pistas	$V_1^\ddagger$	$V_a^\ddagger$	$V_b^\ddagger$	c (veh)	$E[T_1]^\sharp$
a_1	1,50 (2,41)	1	55 (88)	50 (80)	20 (32)	300	0,0273 (98)
a_2	1,00 (1,61)	1	55 (88)	50 (80)	20 (32)	200	0,0182 (65)

Obs.: *em milhas (em km); ‡em milhas por hora (em km/h); $^\sharp$em h (em s);

Pela Tabela 3.3, observamos que quando não há chegadas ($\lambda = 0$), o tempo esperado é igual ao menor tempo de percurso, que corresponde à condição de sistema vazio e velocidade máxima. Ou seja, o ocupante está sozinho no sistema e não tem impedimento de trafegar na velocidade máxima permitida. A partir de $\lambda = 500$, observamos um aumento no tempo esperado de percurso, restrito ao trecho utilizado. Adicionalmente, conforme esperado, o algoritmo direciona o tráfego preferencialmente ao trecho mais rápido, o que é promissor. Notamos também que o tempo de serviço nunca é igual nos dois trechos. Isto significa que um usuário com conhecimento sobre o

Tabela 3.3: Atribuição ótima para rede de dois trechos

λ	rota	atribuição	$E[T]^*$
0	a_1	n/a	0,0273 (98)
	a_2	n/a	0,0182 (65)
500	a_1	0	0,0273 (98)
	a_2	500	0,0190 (68)
1.000	a_1	0	0,0273 (98)
	a_2	1.000	0,0202 (73)
2.000	a_1	408	0,0282 (101)
	a_2	1.592	0,0222 (80)
4.000	a_1	674	0,0290 (104)
	a_2	1.711	0,0228 (82)
8.000	a_1	658	0,0289 (104)
	a_2	1.709	0,0228 (82)

*Tempo em horas (em segundos);

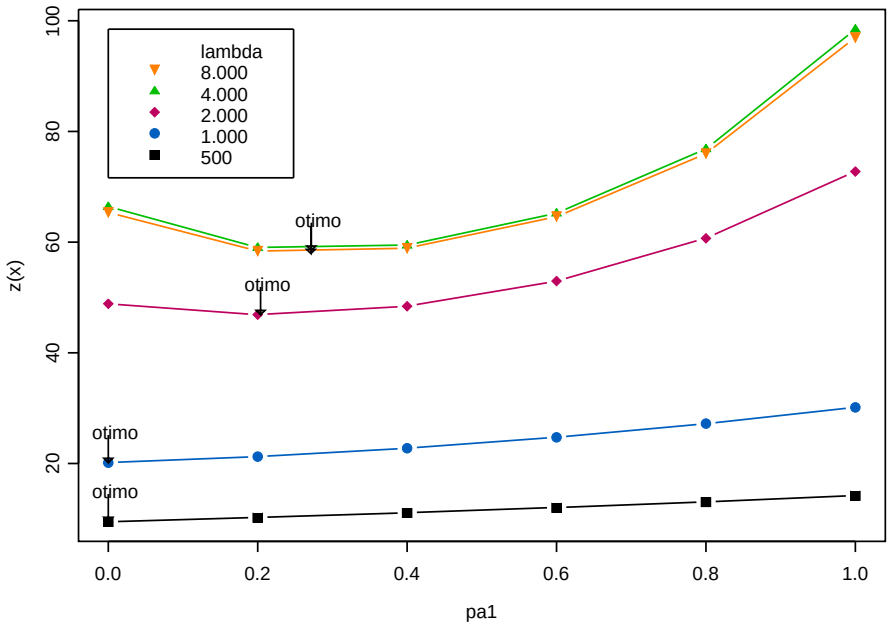

Figura 3.2: Função objetivo e soluções ótimas

sistema poderia reduzir o próprio tempo de percurso de A a B, mudando da rota mais lenta para a mais rápida. Tal melhoria é impossível no ótimo do modelo UE (veja Sheffi, 1985), mas não no do modelo SO. De fato, o modelo SO busca minimizar custos *globais* (e não *individuais*).

Observamos também pela Tabela 3.3 uma saturação da função objetivo, aparentemente a partir de $\lambda \approx 2.400$, uma vez que não há diferenças perceptíveis para a atribuição do tráfego, para taxas de chegada $\lambda > 2.400$. Notamos também que no estado de saturação a rede alcança sua capacidade e a restrição (1.2) não é mais atendida. Este é um importante tópico para investigação futura.

Uma dúvida que havia sido levantada é se o algoritmo DE estaria realmente convergindo para o ótimo global. Na Figura 3.2, verificarmos a acurácia das soluções obtidas, pelos gráficos da função objetivo, Eq. (1.1), para várias taxas de chegada λ.

Entretanto, um resultado surpreendente surgiu, que parece contrariar a conjectura inicial de que o tempo de percurso não-convexo produzirá uma função objetivo também não-convexa. Experimentos adicionais necessitam ser conduzidos, para verificar sob quais condições isto ocorre (topologia de rede, taxas de chegada e assim por diante).

Capítulo 4

Conclusões

4.1 Observações Finais

Tivemos como objetivo nesta dissertação investigar um modelo de atribuição de tráfego de veículos, o clássico modelo do ótimo do sistema (SO, do inglês, *system optimum*). Este é um modelo largamente conhecido dos pesquisadores da área e para o qual há uma diversidade grande de algoritmos para resolução, tanto exatos quanto aproximados. A novidade nesta dissertação foi a consideração de modelos de filas para a estimação do tempo de percurso, a principal medida de desempenho considerada nos modelos SO.

De fato, resolvemos o modelo SO com a utilização de uma expressão para o tempo entre viagens, baseada em filas $M/G/c/c$ dependentes do estado. Esta nova expressão representa com maior precisão o fenômeno de congestionamento (isto é, os tempos de viagem aumentam com o aumento do congestionamento), embora traga dificuldades para o algoritmo de otimização, pela não convexidade que acarreta a possibilidade de ocorrência de ótimos locais.

Obtivemos resultados com uma heurística evolucionária do tipo DE (*Differential Evolution*). Resultados preliminares indicaram que a heurística DE não tem um comportamento explosivo do tempo de processamento em função

do número de variáveis de decisão. Esse é o comportamento esperado de um algoritmo considerado eficiente, pelos pesquisadores da área de desenvolvimento de algoritmos. Os resultados mostram também que as atribuições de tráfego produzidas são coerentes e robustas.

4.2 Propostas de Continuidade

Ao final desta pesquisa, algumas questões foram respondidas, mas muitas outras questões surgiram. Trabalhos futuros poderiam incluir algumas a investigação de algumas destas questões.

Tópicos para trabalhos futuros incluem, por exemplo, a análise de redes de dimensões maiores. Em outras palavras, configurações com maior número de vias e topologias, que incluam fusões e divisões de vias, poderiam ser analisadas. Também é de interesse conhecer o desempenho do algoritmo frente a aplicação reais, com redes de tamanhos e configurações reais.

Uma outra questão relevante é como o algoritmo se comportaria em redes de pedestres. De fato, a principal característica aqui modelada (ou seja, o fenômeno de congestionamento, com a consequente redução da velocidade de percurso em função do aumento do número de usuários) também se aplica ao tráfego de pessoas.

Referências Bibliográficas

Akçelik, R. (1991). Travel time function for transport planning purposes: Davidson's function, its time dependent form and an alternative travel time function, *Australian Road Research* **21**(3): 49–59.

Artalejo, J. R. & Gómez-Corral, A. (2008). *Retrial Queueing Systems: A Computational Approach*, Springer, New York.

Babu, B. V. & Sastry, K. K. N. (1999). Estimation of heat transfer parameters in a trickle bed reactor using differential evolution and orthogonal collocation, *Computers & Chemical Engineering* **23**(3): 327–339.

Bureau of Public Roads (1964). Traffic assignment manual, *Technical Report*, U.S. Department of Commerce.

Cardoso, F. F. (2010). *Problemas de alocação de tráfego sujeitos a congestionamento*, Dissertação de mestrado, Programa de Pós-Graduação em Engenharia Elétrica - EE - UFMG, Belo Horizonte - MG.

Carvalho, G. D., Nascimento, R. A. & Cruz, F. R. B. (2011). Atribuição de tráfego em redes de filas finitas dependentes do estado, *XII Escola de Modelos de Regressão - XII EMR [Anais]*, Associação Brasileira de Estatística, Fortaleza, CE, pp. 1–4.

Cruz, F. R. B., van Woensel, T., Smith, J. M. & Lieckens, K. (2010). On the

system optimum of traffic assignment in $M/G/c/c$ state-dependent queueing networks, *European Journal of Operational Research* **201**(1): 183–193.

Drake, J. S., Schofer, J. L. & May, A. D. (1967). A statistical analysis of speed density hypotheses, *Highway Research Record* **154**: 53–87.

Edie, L. C. (1961). Car following and steady-state theory, *Operations Research* **9**: 66–76.

Fan, H.-Y. & Lampinen, J. (2004). A trigonometric mutation operation to differential evolution, *Journal of Global Optimization* **27**(1): 105–129.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Greenshields, B. D. (1935). A study of traffic capacity, *Highway Research Board Proceedings* **14**: 448–477.

Helbing, D., Schönhof, M., Stark, H.-U. & Holyst, J. A. (2005). How individuals learn to take turns: Emergence of alternating cooperation in a congestion game and the prisoner's dilemma, *Advances in Complex Systems* **8**(1): 87–116.

Hillier, F. S. & Lieberman, G. J. (2005). *Introduction to Operations Research*, 8a ed., McGraw Hill Higher Education.

Jain, R. & Smith, J. M. (1997). Modeling vehicular traffic flow using $M/G/C/C$ state queueing models, *Transportation Science* **31**(4): 324–336.

Lampinen, J. (2000). A bibliography of differential evolution algorithm, *Technical Report*, Laboratory of Information Processing, Department of

Information Technology, Lappeenranta University of Technology. **URL:** *http://www.lut.fi/~jlampine/debiblio.htm*

Lampinen, J. & Zelinka, I. (1999). Mechanical engineering design optimization by differential evolution, *in* D. Corne, M. Dorigo & F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, pp. 127–146.

Price, K. & Storn, R. (2010). Differential evolution, *Website of DE as on August 2010*, International Computer Science Institute, University of California, Berkeley. **URL:** *http://www.icsi.berkeley.edu/~storn/code.html*

Sheffi, Y. (1985). *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*, Prentice-Hall, Englewood Cliffs, NJ.

Storn, R. & Price, K. (1997). Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* **11**(4): 341–359.

Transportation Research Board (1985). Highway capacity manual, *Special Report 209*, National Research Council.

Underwood, R. T. (1961). Speed, volume, and density relationships: Quality and theory of traffic flow, *Yale Bureau of Highway Traffic* pp. 141–188.

Apêndice A

Códigos em C++

Código A.1: mgccso-main.cpp

```
1  //
2  // Purpose:
3  //    to assign traffic in networks of MGCC queues
4  //
5  //
6  // Version:
7  //    3.0
8  //
9  // Date:
10 //    Nov/2010
11 //
12 #include <stdlib.h>
13 #include <stdio.h>
14 #include "functions.c"
15 #include "DESchemes.c"
16 #include "cmusr.cpp"
17 #include "mgccso.cpp"
18 int main(int argc, char *argv[]) {
19   // check input
20   if (argc < 2) {
21      fprintf(stderr, "mgccso-main:\t assigns traffic in MGcc networks\n");
22      fprintf(stderr, "Usage:\t mgccso-main [script_file]\n");
23      exit(0);
24   }
25   OpenDEpar();
26   int aux = 0;
27   while ((argv[1][aux]!='\n')&&(argv[1][aux]!='\0')) aux++;
28   argv[1][aux] = '\0';
29   FILE *inputFile = fopen(argv[1],'r');
30   if (inputFile == NULL) {
31         fprintf(stderr, "%s: No such file\n", argv[1]);
32       exit(0);
33   }
34   MgccNet myMgccNet;
35   myMgccNet.ReadData(inputFile);
36   myMgccNet.ShowNet();
37   const int LLENGTH = 256;
38   char Line[LLENGTH];
39   int i;
40   fgets(Line, LLENGTH, inputFile);
41   fprintf(stdout,"Final Results:\n");
42   determineDV();
43   createSpecimen(nPop);
44   PrepareSpecimen();
45   createHistory(Generations);
46   createPopulation();
47   createTMPPopulation();
48   SetScheme();
49   InitializePopulation();
50   (void) time(&t1);
51   nmut=0;
```

```
52      neval=0;
53      //initialize population
54      srand((unsigned)time(NULL));
55      srand(135790);
56      for(a1=0;a1<PNP-3;a1++) {
57              for(a2=0;a2<PD-1;a2++) {
58                  setPopulation(a2, a1,
59                          getSpecimen(0,a2)+
60                          ((double)rand()/RAND_MAX)*(getSpecimen(1,a2) - getSpecimen(0,a2))
61               );
62              }
63      }
64      for(l=0;l<Generations;l++) {
65              //evolution
66              for(a1=0;a1<PNP-3;a1++) {
67                  SetProb(a1,0);
68               myMgccNet.GetPerfMeasures(0);
69               myMgccNet.AggregatePerfMeasures();
70               setPopulation(PD-1,a1,sumd1);
71               neval++;
72               if(Scheme==1) {
73                      DER1B();
74                  } else {
75                      if(Scheme==2) {
76                          DER2B();
77                      } else {
78                          DECuRR();
79                      }
80               }
81               //best individual into new generation
82                  SetProb(PNP-3,1);
83                  int origin, dest;
84                  change=0;
85                  conNew=0;
86                  conOld=0;
87               sumNew=0;
88                  sumOld=0;
89               for(g=0;g<iKT;g++) {
90                              origin=ArcKT[g][0];
91                              dest=ArcKT[g][1];
92                              if(ArcKT[g][0]==ArcKT[g+1][0]) {
93                                  sumNew+=fabs(prob[1][origin-1][dest-1]);
94                                  sumOld+=fabs(prob[0][origin-1][dest-1]);
95                              } else {
96                                  if(prob[1][origin-1][dest-1]!=1) {
97                                      sumNew+=fabs(prob[1][origin-1][dest-1]);
98                                      sumOld+=fabs(prob[0][origin-1][dest-1]);
99                                      sumNew--;
100                                     sumOld--;
101                                     maxNew=maximum(sumNew,0);
102                                     maxOld=maximum(sumOld,0);
103                                     if(maxNew < maxOld) {
104                                         con1++;
105                                         conNew+=maxNew;
106                                         conOld+=maxOld;
107                                     }
108                                     sumNew=0;
109                                     sumOld=0;
110                                 }
111                             }
112                 }
113              if(conNew > 0) {
114                              change=1;
115                  } else {
116                              if(conOld>0) {
117                                  change=1;
118                                  myMgccNet.GetPerfMeasures(1); //New
119                                  myMgccNet.AggregatePerfMeasures();
120                                  setPopulation(PD-1,PNP-3,sumd1);
121                                  neval++;
122                              } else {
123                                  myMgccNet.GetPerfMeasures(1); //New
124                                  myMgccNet.AggregatePerfMeasures();
125                                  setPopulation(PD-1,PNP-3,sumd1);
126                                  neval++;
127                                  if(getPopulation(PD-1,a1)==0) {
128                                      myMgccNet.GetPerfMeasures(0); //Old
129                                      myMgccNet.AggregatePerfMeasures();
130                                      setPopulation(PD-1,a1,sumd1);
131                                      neval++;
132                                  }
133                                  if(getPopulation(PD-1,PNP-3)≤getPopulation(PD-1,a1)) {
134                                      change=1;
```

```
                                }
                            }
                        }
                        if(change==1) {
                            nmut++;
                            for(a3=0;a3<PD;a3++) {
                            setTMPPopulation(a3,a1,getPopulation(a3,PNP-3));
                            }
                        } else {
                        for(a3=0;a3<PD;a3++) {
                            setTMPPopulation(a3,a1,getPopulation(a3,a1));
                        }
                        }
                        setPopulation(PD-1,PNP-3,0);
                    }
                    //new population replace old population
                for(a1=0;a1<PNP;a1++) {
                    for(a2=0;a2<PD;a2++) {
                        setPopulation(a2,a1,getTMPPopulation(a2,a1));
                     }
                  }
                    //search for the best individual
                    WorstIndividual();
                    BestIndividual();
                    if (hh-h≥Stagnation[0]-Precision*(hh-h) &&
                        hh-h≤Stagnation[0]+Precision*(hh-h)) {
                        Stagnation[1]++;
                    } else {
                        Stagnation[0]=h-hh;
                        Stagnation[1]=0;
                    }
                    //append to History
/*
                    system("cls");
                    printf("Scheme = %d\n", Scheme);
                    printf("Generation = %d\n", l+1);
                    printf("Population\n");
                    for (a1=0;a1<PNP-3;a1++) {
                     for (a2=0;a2<PD;a2++) {
                        printf("%lf\t", Population[PD * a1 + a2]);
                     }
                     printf("\n");
                  }
                    printf("Best individual = %lg\n", h);
                    printf("Worst individual = %lg\n", hh);
*/
                    History[l]=h;
                    ldef=l;
                    if(h != 0 && hh != 0) {
                        if( ((hh-h)/fabs(hh)<Precision)||(Stagnation[1]>10000) ) {
                            goto stoprun1;
                        }
                    } else {
                        if(h == 0) {
                            Stagnation[2]++;
                            if(Stagnation[2]>1000000) {
                                goto stoprun1;
                            }
                        }
                    }
                }//Generations
                stoprun1:
                (void) time(&t2);
                WritePOP(MutScheme);
                SetProb(b,0);
                WriteOut(MutScheme);
                WriteHistory();
                destroyTMPPopulation();
                destroyPopulation();
                destroyHistory();
                destroySpecimen();
                while (fscanf(inputFile, "%d/n", &i) == 1 ) {
                    myMgccNet.ShowPerfMeasures(i-1);
                }
                fclose(inputFile);
                return 0;
            }
```