Joaquim Manuel Pereira Mateus

UM AUTÓMATO CELULAR PARA TRÁFEGO EM MULTIFAIXAS

Departamento de Matemática Aplicada
Faculdade de Ciências da Universidade do Porto
Julho de 2001

Joaquim Manuel Pereira Mateus

UM AUTOMATO CELULAR PARA TRÁFEGO EM MULTIFAIXAS

Dissertação do mestrado em Matemática Aplicada
apresentada para a obtenção do grau de Mestre.

Trabalho realizado sob orientação do
Prof. Doutor Miguel Nunes da Silva.

Departamento de Matemática Aplicada
Faculdade de Ciências da Universidade do Porto
Julho de 2001

Aos meus Pais e Irmão

Ao terminar este trabalho gostaria de agradecer a todos os que, de alguma forma, contribuiram para a sua realização.

Em primeiro lugar, quero agradecer ao Prof. Doutor Miguel Nunes da Silva pela sua disponibilidade, paciência demonstrada e motivação transmitida ao longo deste trabalho. E também, ao Prof. Doutor José Duarte pela sua colaboração na fase inicial deste trabalho.

Agradeço, de uma forma muito especial, à Adelaide por todo o apoio que me deu.

Aos meus colegas do Departamento de Matemática da ESTG do Instituto Politécnico da Guarda porque sempre me apoiaram e incentivaram até ao fim.

Gostaria também de agradecer à minha família e amigos pelo carinho, incentivo e apoio incondicionais mesmo nos momentos mais difíceis.

Resumo

O problema do tráfego automóvel desde há algum tempo que se tornou muito importante, sendo por esse motivo objecto de estudo. Neste trabalho simulam-se computacionalmente modelos de alguns tipos de tráfego, de forma a obter resultados de acordo com a realidade. A importância destas simulações traduz-se no fato de podermos obter resultados e conclusões de uma forma mais rápida e economica do que usando métodos que usassem situações reais. Todos os modelos se baseiam no modelo de autómatos celulares de Nagel-Schreckenberg. O resultado principal deste estudo é o gráfico fundamental do fluxo versus densidade. O modelo mais simples foi inicialmente implementado usando o software Matlab, com o aumento da complexidade dos modelos e a necessidade de um menor tempo de execução do programa, este teve que ser implementado usando a linguagem de programação C, uma vez que desta forma o programa pode ser compilado, o compilador usado foi o lcc-win, como a forma mais "clara" de visualizar os resultados é a gráfica, tornou-se necessário o uso de software gráfico, pelo que usámos o Dislin (Dislcc). Os resultados obtidos, sempre que comparados com os publicados em artigos, são semelhantes. Embora se apresentem resultados originais. A implementação pode ser adaptada a novas situações e a outros tipos de tráfego que introduzirão novas variantes nos modelos, as quais usualmente se designam por regras de tráfego automóvel. Esta implementação tem uma forte componente probabilística, cujos valores dependerão do tipo de condutores e de tráfego que é objecto de estudo.

Abstract

The problem of the traffic flow since has some time that became very important, being for this reason study object. In this work models of some types of traffic are simulated computational, of form to get results in accordance with the reality. The importance of these simulations is expressed the fact to be able to get results and conclusions of a faster and economic way of that using methods that used real situations. All the models are based on the model of cellular automatos of Nagel-Schreckenberg. The main result of this study is the fundamental graph of the flow versus density. Initially, the simplest model was implemented using Matlab software, with the increase in the complexity of the models and the necessity of a higher speed in program execution, this had that to be implemented using the programming language C, in this form the program can be compiled, the used compiler was lcc-win. A "clear" form to visualize the results was the graphical, because of that was necessary a graph software, we used the Dislin (Dislcc). The gotten results are similar with others in published. Although we present original results. The implementation can be customized to new situations and other types of traffic that will introduce new variants in the models, that usually assign for traffic flow rules. This implementation has one strong probabilistic component, whose values will depend on the type of conductors and traffic that is study object.

Índice

Capítulo 1

Introdução

O problema do tráfego automóvel desde há algum tempo que se tornou muito importante, sendo por esse motivo objecto de estudo. Como podemos observar nos nossos dias, mais nas grandes cidades mas não só, o problema do tráfego automóvel é diário. Neste trabalho simulam-se computacionalmente modelos de alguns tipos de tráfego, de forma a obter resultados de acordo com a realidade.

A importância destas simulações traduz-se no facto de podermos obter resultados e conclusões de uma forma mais rápida e economica do que usando métodos que usassem situações reais. Todos os modelos se baseiam no modelo de automatos celulares de Nagel-Schreckenberg. O resultado principal deste estudo é o gráfico fundamental do fluxo versus densidade.

O modelo mais simples, o de uma estrada com uma faixa, onde portanto não existem ultrapassagens, mas apenas movimento de veículos, foi inicialmente implementado usando o software Matlab, com o aumento da complexidade dos modelos e a necessidade de uma maior rapidez de execução do programa, este teve que ser implementado usando a linguagem de programação C, uma vez que desta forma o programa pode ser compilado, o compilador usado foi o lcc-win, como a forma mais " clara " de visualizar os resultados era a gráfica tornou-se necessário o uso de software gráfico para se conseguir esse fim, o software usado foi o Dislin (Dislcc).

Para a implementação do modelo mais simples, ou seja de uma faixa, usaram-se as regras básicas que determinam o movimento de veículos, essas regras são quatro: a primeira que aumenta a velocidade do veículo caso ela não seja a máxima, a segunda regra efextua travagem caso seja necessário, a terceira regra é uma regra probabilistica que pretende reflectir comportamnetos do condutor, como seja por exemplo uma distracção que o leve a reduzir a velocidade e por último uma regra que efectua o movimento dos veículos na faixa de rodagem.

Inicialmente o tráfego em multifaixas não era objecto de estudo das teorias tradicionais, obviamente que na actualidade esse tipo de tráfego é muito importante, uma vez que as vias se tornam cada vez mais complexas e com mais faixas, neste trabalho pretende-se ainda efectuar um estudo e simulação de um modelo que se torna ainda mais complexo do que o modelo de tráfego em multifaixas em auto-estradas[1], esse modelo é referente ao tráfego urbano que como óbviamente se deduz se torna muita mais complexo, por todas as especificidades que tem. Em qualquer caso que se refira multifaixa, há regras a definir para mudança de faixa. No tráfego urbano essas regras têm que contar com outros tipos de motivações que não existem em auto-estradas.

Apresentam-se estudos e resultados menos estudados, como é o caso do modelo em que se entra com dois tipos de veiculos, veículos esses que têm comprimentos e velocidades diferentes, na prática representarão veículos ligeiro e pesados.

De notar que a implementação tem uma forte componente probabilistica, cujos valores dependerão do tipo de condutores e de tráfego que é objecto de estudo.

[1] Entenda-se por auto-estrada: estrada fora do circuito urbano, com mais do que uma faixa de rodagem e onde os sentidos de deslocamento dos veículos estão separados, ou seja, os veículos não podem usar a mesma faixa de rodagem para sentidos diferentes.

Capítulo 2

Preliminares

2.1 Noções Básicas

As noções principais usadas neste trabalho são as de densidade e de fluxo, para além de outras que apresentaremos a seguir:

Definição 2.1 ***Sítio da estrada*** *é o espaço numa estrada correspondente ao comprimento médio de um veículo ligeiro mais o espaço entre veículos num engarrafamento(aproximadamente 7,5 metros).*

Definição 2.2 ***Intervalo de tempo*** *é interpretado como o tempo de reacção humana e toma o valor de 1 segundo.*

Definição 2.3 ***Densidade*** *é o quociente entre o número de veículos na estrada e o número total de sítios da estrada.*

Definição 2.4 ***Fluxo*** *é o quociente entre o número de veículos que passam num determinado sítio da estrada num dado intervalo de tempo e valor desse intervalo de tempo.*

Definição 2.5 $\mathbf{V_{max}}$ *é a abreviatura para velocidade máxima de um veículo e é representada por um número inteiro (por exemplo 5, que pode respresentar a velocidade real de 150 Km/h).*

2.2 Teoria do Campo Médio

A Teoria de campo-médio é um método analítico, que ignora completamente as correlações [2].

A aproximação analítica simples ao modelo NaSch é uma Teoria de campo-médio (microscópica). Aqui consideramos a densidade $c_v(j,t)$ de veículos com velocidade v no sítio j e no tempo t. Na aproximação de campo-médio, as correlações entre sítios são completamente negligenciadas.

Para $v_{\max} = 1$ as equações do campo-médio para o estado estacionário ($t \to \infty$) são:

$$\begin{aligned} c_0 &= (c + pd)c \\ c_1 &= \overline{p}cd \end{aligned} \tag{2.1}$$

com $c = c_0 + c_1$, $d = 1 - c$ e $\overline{p} = 1 - p$. O fluxo é simplesmente dado por $f_{MF}(c) = c_1$.

Para dinâmicas de sequência-aleatória (em cada intervalo de tempo cada sítio é actualizado aleatoriamente) a aproximação de campo-médio é conhecida por ser exacta para $v_{\max} = 1$. Para dinâmicas paralelas, contudo, a teoria de campo-médio subestima consideravelmente o fluxo.

Para $v_{\max} = 2$ as equações de relação para as densidades são dadas por

$$\begin{aligned} c_0 &= (c+pd)c_0 + (1+pd)c(c_1+c_2) \\ c_1 &= d\left[\overline{p}c_0 + (\overline{p}c + pd)\,(c_1+c_2)\right] \\ c_2 &= \overline{p}d^2\,(c_1+c_2) \end{aligned} \tag{2.2}$$

A solução é dada por

$$\begin{aligned} c_0 &= \frac{(1+pd)\,c^2}{1-pd^2} \\ c_1 &= \frac{\overline{p}\,(1-\overline{p}d^2)\,dc}{1-pd^2} \\ c_2 &= \frac{\overline{p}^2 d^3 c}{1-pd^2} \end{aligned} \tag{2.3}$$

e o fluxo pode ser calculado usando $f_{MF}(c) = c_1 + 2c_2$. Mais uma vez o fluxo é subestimado consideravelmente. Isto é verdade para $v_{\max}$ arbitrário.

A solução analítica exacta do modelo de tráfego não é possível excepto para o caso da velocidade máxima $v_{\max} = 1$ [1]. Por esse motivo, algumas aproximações são necessárias quando se tenta estudar este modelo analiticamente para $v_{\max} > 1$. O seguinte tipo de teoria de campo-médio será o primeiro passo para a tal aproximação analítica.

O cálculo começa pela descrição estocástica do modelo de tráfego. Em vez de se especificar a posição dos veículos e as suas velocidades, analisa-se a distribuição probabilística de cada

sítio em cada passo de tempo. Representa-se a probabilidade do sítio $i\,(i = 1, 2, 3, ..., L)$ não ter veículo no instante de tempo t por $d\,(i,t)$ e a probabilidade do sítio i ter um veículo com velocidade $\alpha\,(\alpha = 0, 1, 2, ..., v_{\max})$ no instante de tempo t por $c_\alpha\,(i,t)$. As distribuições c e d, juntas, têm em conta todos os estados possíveis dos diferentes sítios. Deste modo temos a condição de normalização para todos os sítios em todos os intervalos de tempo

$$d\,(i,t) + c_0\,(i,t) + c_1\,(i,t) + c_3\,(i,t) + ... + c_{v_{\max}}\,(i,t) = 1 \tag{2.4}$$

Representando $c\,(i,t)$ a probabilidade total para o sítio i estar ocupado no instante de tempo t, i.e., $\sum_{j=0}^{v_{\max}} c_j\,(i,t)$, temos apenas $d\,(i,t) + c\,(i,t) = 1$.

De acordo com as regras de actualização em quatro estados a evolução do tempo dessas distribuições probabílisticas pode ser descrita pelo seguinte conjunto de quatro equações:

- Estado de Aceleração

$$\begin{aligned} c_0\,(i,t_1) &= 0 \\ c_\alpha\,(i,t_1) &= c_{\alpha-1}\,(i,t)\,,\ 0 < \alpha < v_{\max} \\ c_{v_{\max}}\,(i,t_1) &= c_{v_{\max}}\,(i,t) + c_{v_{\max}-1}\,(i,t) \end{aligned} \tag{2.5}$$

- Estado de Travagem

$$c_0\,(i,t_2) = c_0\,(i,t_1) + c\,(i+1,t_1) \sum_{\beta=1}^{v_{\max}} c_\beta\,(i,t_1) \tag{2.}$$

$$c_\alpha\,(i,t_2) = c\,(i+\alpha+1,t_1) \prod_{j=1}^{\alpha} d\,(i+j,t_1) \sum_{\beta=\alpha+1}^{v_{\max}} c_\beta\,(i,t_1) + c_\alpha\,(i,t_1) \prod_{j=1}^{\alpha} d\,(i+j,t_1)\,,\ 0 < \alpha < v_{\max}$$

$$c_{v_{\max}}\,(i,t_2) = \prod_{j=1}^{v_{\max}} d\,(i+j,t_1)\, c_{v_{\max}}\,(i,t_1)$$

- Estado de Aleatoridade

$$
\begin{aligned}
c_0(i,t_3) &= c_0(i,t_2) + pc_1(i,t_2) \\
c_\alpha(i,t_3) &= qc_\alpha(i,t_2) + pc_{\alpha+1}(i,t_2),\ 0 < \alpha < v_{\max} \\
c_{v_{\max}}(i,t_3) &= qc_{v_{\max}}(i,t_2)
\end{aligned}
\tag{2.7}
$$

- Estado de Movimento

$$
c_\alpha(i,t+1) = c_\alpha(i-\alpha,t_3),\ 0 \le \alpha \le v_{\max} \tag{2.8}
$$

Assume-se que o tempo t assume apenas valores inteiros e os parâmetros t_1, t_2 e t_3 representam os passos intermédios de tempo entre t e $t+1$ (por vezes definidos como $t+1/4$, $t+1/2$ e $t+3/4$, respectivamente). Estas equações de evolução ao longo do tempo conservam, para condições de fronteira periódicas, o número total de veículos em cada estado. Esta formulação da dinamica negligencia por completo as correlações espaciais uma vez que assumimos que todos os valores expectativos são factorizados em termos locais.

As variáveis c e d tomam valores reais entre 0 e 1. A descrição estocástica provém da natureza estocástica do terceiro passo aleatório. Os outros três passos são puramente determinísticos.

Estas equações de evolução ao longo do tempo são não-lineares e a análise completa de todo o sistema não tem tido sucesso até ao momento. Contudo, no estado estacionário, i.e. no limite $t \to \infty$, as distribuições c e d tornam-se homogéneas no espaço (para condições de fronteira periódicas) e assim, à parte da dependência do tempo, também a dependência do sítio pode ser omitida. Usando estas considerações e combinando os quatro passos de actualização obtemos o seguinte conjunto de $v_{\max}+1$ equações:

$$
\begin{aligned}
c_0 &= (c+pd)\,c_0 + (1+pd)\,c\sum_{\beta=1}^{v_{\max}} c_\beta \\
c_\alpha &= d^\alpha\left[qc_{\alpha-1} + (qc+pd)\,c_\alpha + (q+pd)\,c\sum_{\beta=\alpha+1}^{v_{\max}} c_\beta\right],\ 0 < \alpha < v_{\max}-1 \\
c_{v_{\max}-1} &= d^{v_{\max}-1}\left[qc_{v_{\max}-2} + (qc+pd)\,(c_{v_{\max}-1} + c_{v_{\max}})\right] \\
c_{v_{\max}} &= qd^{v_{\max}}\left[c_{v_{\max}-1} + c_{v_{\max}}\right]
\end{aligned}
\tag{2.9}
$$

Essencialmente, estas equações descrevem o movimento de um único veículo com "obstácu-

los" estatísticos (dependente da densidade). Uma característica interessante destas equações é que elas são lineares quando especificamos a densidade $c = 1 - d$ de veículos. Assim as equações (2.9) podem ser reformuladas em notação matricial como $A\overrightarrow{c} = \overrightarrow{c}$. A matriz A pode ser obtida a partir de (2.9), $\overrightarrow{c}$ é o vector com elementos c_α, $\alpha = 0, ..., v_{\max}$. Para baixos valores de $v_{\max}$ podemos inverter a matriz A para encontrar explicitamente o valor das densidades c_α. Para altos valores de $v_{\max}$ torna-se mais conveniente escrever uma relação recursiva de forma a obter a solução do estado estável.

Da primeira equação em (2.9) podemos determinar c_0 directamente sem conhecer os outros $c_\alpha^{'}s$ e obter

$$c_0 = c^2 \frac{1 + pd}{1 - pd^2} \tag{2.10}$$

Usando este resultado e a segunda equação de (2.9) podemos também escrever a expressão para c_1

$$c_1 = qc^2 d \frac{1 + d + pd^2}{(1 - pd^3)(1 - pd^2)} \tag{2.11}$$

Para α maior que um, uma relação recursiva pode ser deduzida calculando $c_\alpha - dc_{\alpha-1}$ usando novamente a segunda equação de (2.9)

$$c_\alpha = \frac{1 + (q - p)\, d^\alpha}{1 - pd^{\alpha+2}} dc_{\alpha-1} - \frac{qd^\alpha}{1 - pd^{\alpha+2}} c_{\alpha-2} \tag{2.12}$$

para $\alpha = 2, 3, ..., v_{\max} - 2$. Assim, começando com as expressões (2.10) e (2.11) para c_0 e c_1, podemos estimar recursivamente c_2, c_3, ..., $c_{v_{\max}-2}$. Estes resultados não dependem do valor actual de $v_{\max}$ e assim são válidos genericamente (desde que $v_{\max} \geq 2$).

Finalmente, para as duas ultimas equações de (2.9) obtemos

$$c_{v_{\max}-1} = \frac{1 - qd^{v_{\max}}}{1 - d^{v_{\max}-1}(q + pd)} qd^{v_{\max}-1} c_{v_{\max}-2} \tag{2.13}$$

$$c_{v_{\max}} = \frac{qd^{v_{\max}}}{1 - qd^{v_{\max}}} c_{v_{\max}-1} \tag{2.14}$$

A dependência de $v_{\max}$ só ocorre nas duas últimas equações. "Referencia a uma figura com algumas densidades para valores elevados de vmax". A densidade dos veículos com velocidades elevadas tende para zero rapidamente, uma vez que esperamos uma descida exponencial rápida para as relações recursivas (2.12), (2.13) e (2.14). A contribuiçao dos veículos com altas ve-

locidades são por isso neglegenciadas. Estaremos principalmente interessados no fluxo $f(c,p)$, definido por

$$f(c,p) = \sum_{\alpha=1}^{v_{\max}} \alpha c_\alpha \tag{2.15}$$

No limite, quando $v_{\max} \to \infty$ é possível continuar com a análise usando as equações iterativas (2.10), (2.11) e (2.12) e a equação geradora

$$g(z) = \sum_{\alpha=0}^{\infty} z^\alpha c_\alpha \tag{2.16}$$

Usualmente simplifica-se fazendo $g'(1) = f(c,p)$. As equações (2.10), (2.11) e (2.12) podem agora ser associadas de forma a termos uma única equação para $g(z)$

$$g(z)[1-dz] - g(dz)\,d^2(1-z)[p+qz] = c^2 + pc^2 d(1-z) \tag{2.17}$$

Uma vez que $v_{\max}$ é infinito não temos de nos preocupar com as equações de fronteira superiores (2.13) e (2.14) para $c_{v_{\max}-1}$ e $c_{v_{\max}}$ cuja contribuição é neglegenciada. Notemos que a função geradora g surge com dois argumentos diferentes (z e dz) o que torna impossível de resolver esta equação (linear) explicitamente para $g(z)$.

Depois da diferenciação a expressão para $g'(1)$ é obtida

$$g'(1) = d(1-pc) - g(d)\frac{d^2}{c} \tag{2.18}$$

Por isso o problema reduz-se a encontrar uma expressão para $g(d)$. Esta pode ser encontrada por alicação sucessiva da equação 2.17 com $z = d^n, n = 1,2,3,....$O resultado final é uma expressão assimptótica para $g'(1) = f(c,p)$

$$f(c,p) = qcd\left[1 + \sum_{n=1}^{\infty} d^{2n} \prod_{l=0}^{n-1}\left(p + qd^l\right)\right] \tag{2.19}$$

" Referência a uma figura onde o fluxo f é apesentado em função da concentração c dos veículos para vários valores de desacelaração probabílistica p". O declive na origem (c~0) do diagrama fundamental é infinito enquanto que o declive para c~1 é $-q$. Este resultado de campo-médio produz, comparado com a simulação de dados mostrada acima, valores demasiado pequenos do fluxo. Isto pode ser facilmente entendido uma vez que a redução do problema a um único veículo ignora todas as correlações espaciais dos veículos. Veículos com altas velocidades tendem a ficar equidistantes e podem manter uma velocidade elevada com uma probabilidade

mais elevada que no sistema de campo-médio onde é muito mais difícil de acelerar e mnter-se com uma velocidade elevada durante um certo período.

Além disso, mesmo para $v_{max} = 1$, as soluções de campo-médio não coincidem com o resultado exacto obtido através da teoria de campo-médio melhorada (tradução de "Improved mean-field Theory"). De notar que para actualizações sequenciais aleatórias a solução de campo-médio é exacta para $v_{max} = 1$.

Capítulo 3

O Modelo de 1 Faixa

Antes de estabelecermos os detalhes do modelo e as regras, vamos fixar algumas considerações. O espaço, o tempo e a velocidade são grandezas discretas, cada sítio ou está ocupado por um veículo ou está livre. A velocidade é considerada no intervalo 0,...,$V_{\max}$.

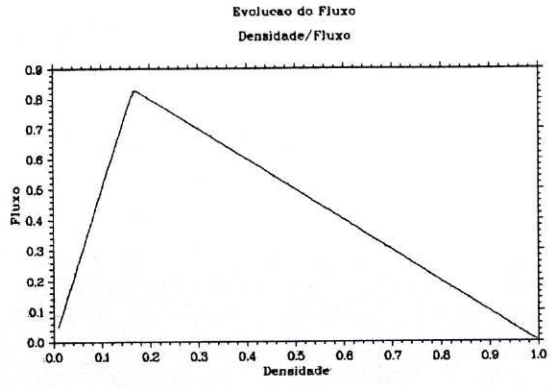
Vamos denotar por n o intervalo de tempo corrente, por $(\Delta x)_n$ a distância entre o veículo em estudo e o que segue à sua frente (distância entre as frentes dos dois veículos), por v_n e x_n a velocidade e posição actual, respectivamente. Então temos o seguinte conjunto de regras, que são actualizadas em simultâneo para todos os veículos:

1) $v = \min(v_n + 1, (\Delta x)_n - 1, v_{\max})$

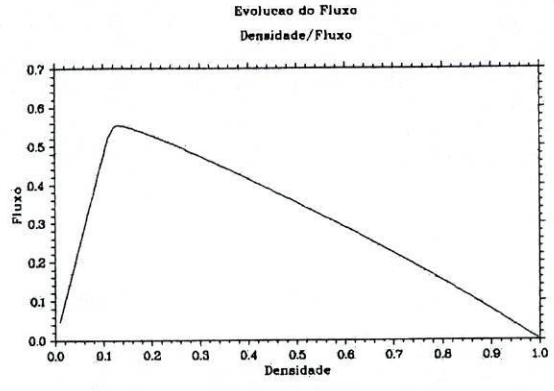
2) $v_{n+1} = \begin{cases} \max(0, v-1) & \text{com probabilidade P\_trava} \\ v & \text{com probabilidade 1-P\_Trava} \end{cases}$

3) $x_{n+1} = x_n + v_{n+1}$

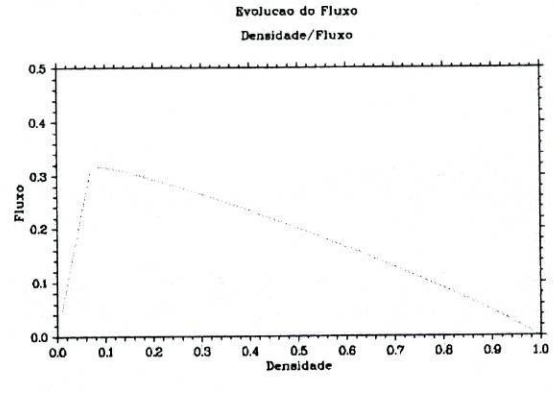
Com a implementação do modelo segundo estas regras obtemos os seguintes resultados, apresentados nos seguintes gráficos:



Figura~3-1: Gráfico obtido para estrada com 10000 sítios e uma probabilidade de reduzir a velocidade de 0.0



Figura~3-2: Gráfico obtido para estrada com 10000 sítios e uma probabilidade de reduzir a velocidade de 0.2



Figura~3-3: Gráfico obtido para estrada com 10000 sítios e uma probabilidade de reduzir a velocidade de 0.5

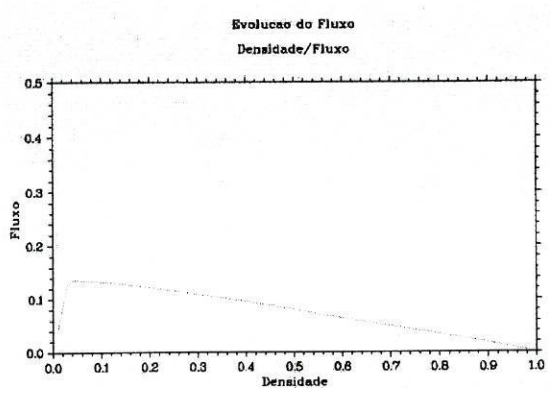


Figura 3-4: Gráfico obtido para estrada com 10000 sítios e uma probabilidade de reduzir a velocidade de 0.8

Capítulo 4

O Modelo de 2 Faixas

Este modelo baseia-se no anterior embora neste caso existam duas faixas, uma situação típica de auto-estrada. Por esse facto há a considerar uma caracteristica neste tipo de tráfego que é a mudança de faixa, tanto da direita para a esquerda como vice-versa.

Neste caso, a actualização tem que ser dividida em duas partes, a primeira é a mudança de faixa e a segunda o movimento dos veículos.

Para evitar determinadas situações desagradáveis, convém ser-se sensível a algumas questões de segurança, i.e., não é permitido a um veículo que pretende mudar de faixa embaraçar o veículo que vem atrás na outra faixa. O que significa que, o veículo que pretende mudar de faixa, deve verificar o seguinte, em relação ao veículo que vem atrás na outra faixa (denotamos por "b" esse veículo e por "o" a outra faixa):

4) $v_{\max}^{o,b} \leq \Delta x^{o,b} - 1$ (onde $v_{\max}^{o,b}$ é a velocidade máxima do veículo que vem atrás na outra faixa e $\Delta x^{o,b}$ é a distância desse mesmo veículo ao veículo que pretende mudar de faixa).

Esta condição é necessária para haver mudança de faixa, mas não suficiente. Existe um outro conjunto de regras que descreve qual o veículo que efectivamente pretende mudar de faixa. Este conjunto de regras é definido de forma a que o veículo tenha que travar o menos possível.

Considerando primeiro a mudança de faixa da direita para a esquerda, usa-se o seguinte conjunto de regras:

5) $v_{\max} > \Delta x - 1$ (não é permitido ir tão rápido como desejava)

e

6) $\Delta x^o \geq \Delta x$ (a outra faixa não é pior), Δx^o é distância ao veículo que vai à frente na outra faixa.

O que significa que mudamos de faixa se estivermos impedidos na faixa corrente e a faixa

da esquerda tiver mais espaço.

A mudança da faixa da esquerda para a direita é realizada, desde que haja espaço livre tanto na faixa da direita como na da esquerda, i.e.:

7) $v_{\max} < \Delta x - 1 - v_{red}$ (existe espaço livre na faixa da esquerda)

e

8) $v_{\max} < \Delta x^o - 1 - v_{red}$ (existe espaço livre na faixa da direita)

Nestas duas regras introduzimos um novo parâmetro, v_{red} , que reduz a distância considerada inicialmente ($\Delta x - 1$ e $\Delta x^o - 1$), e desta forma quanto maior for este parâmetro, maior tem que ser o espaço livre em ambas as faixas, para a mudança da faixa da esquerda para a direita ser possível. É comum observar-se que a faixa da esquerda reservada às ultrapassagens é muitas vezes mais utilizada que a faixa da direita. É óbvio que este comportamento leva a que não se atinga o fluxo máximo possível na faixa da direita. Este comportamento deve-se muitas vezes ao facto de o condutor temer encontrar na faixa da direita um veículo lento e por esse facto muda de faixa cedo demais. Isto traz desvantagens e para grandes densidades de tráfego, as regras apresentadas levam a um congestionamento completo da faixa da esquerda, deixando a faixa da direita livre. Perante esta situação verifica-se que a regra 7) não pode ser cumprida, porque desse modo o regresso à faixa da direita não seria possível.

Com o objectivo de corrigir este efeito, um novo conjunto de regras tem que ser introduzido. Este novo conjunto é simplesmente a regra de segurança 4) e a regra 8) com uma pequena alteração, mas omitindo a regra 7):

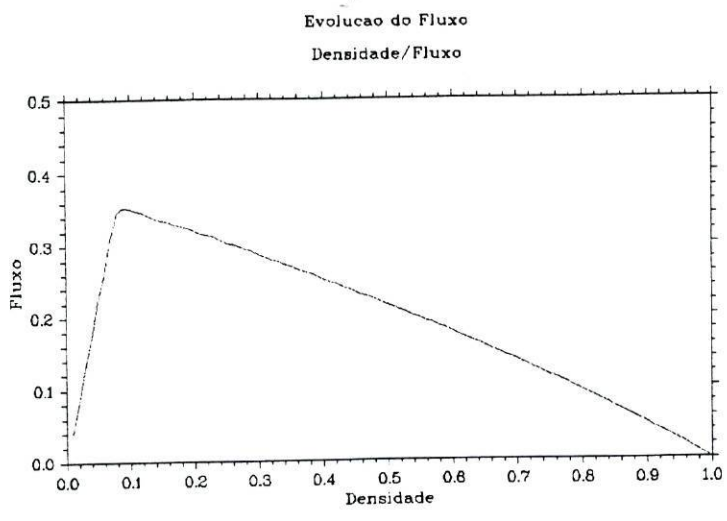
9) $v_{\max}^{o,b} \leq \Delta x^{o,b} - 1$ (para o veículo que vem atrás na faixa da direita)

e

10) $v \leq \Delta x^o - 1$ (espaço livre na faixa da direita)

Desta forma obtivemos um modelo com regras que permitem efectuar mudança de faixa e controlar o movimento dos veículos, para tráfego e multifaixas. Embora o mais usual seja considerarem-se duas faixas, este modelo pode usar-se para um número superior.

Como resultado deste modelo, apresentamos o seguinte gráfico:

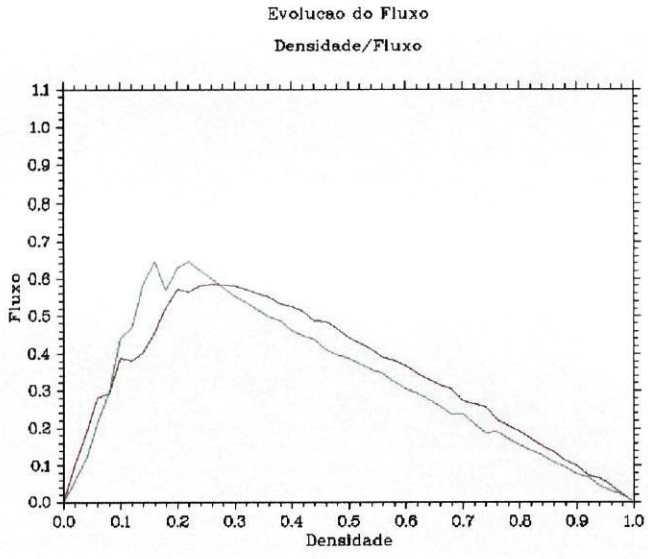

Evolucao do Fluxo
Densidade/Fluxo
Fluxo
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Densidade

Figura~4-1:

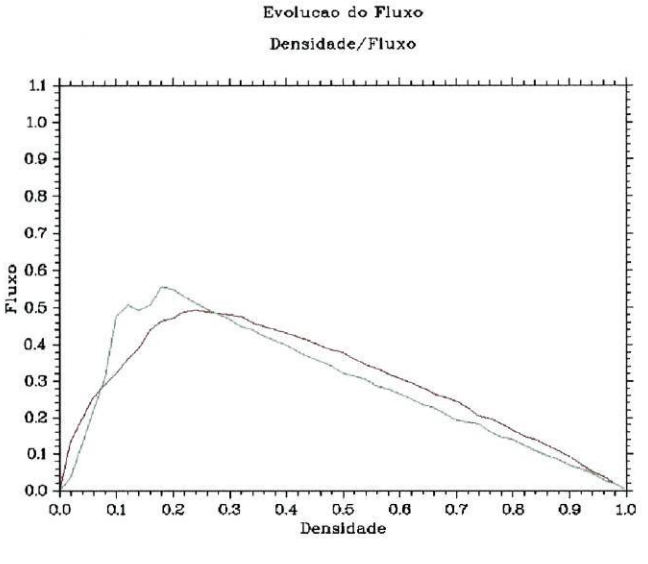
Capítulo 5

O Modelo de 2 Faixas com 2 Tipos de Veículos

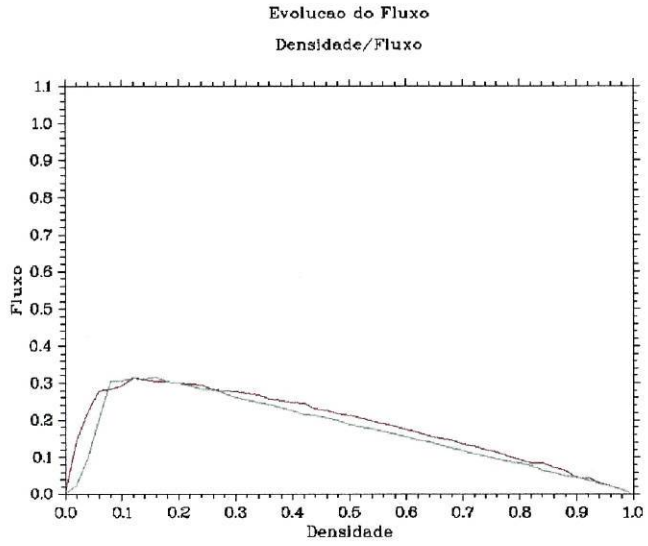
Este modelo obtém-se tendo como base o anterior, mas considerando dois tipos de veículos, designados por ligeiros e pesados. Para a simulação do resultado consideramos que a estrada tem 15% de veículos pesados e 85% de veículos ligeiros, e ainda que $V_{\max}$ dos pesados é igual a 4 e dos ligeiros 6, ocupando cada veículo ligeiro um sítio da estrada e um pesado 3 sítios. Exemplos de resultaos obtidos são os seguintes:



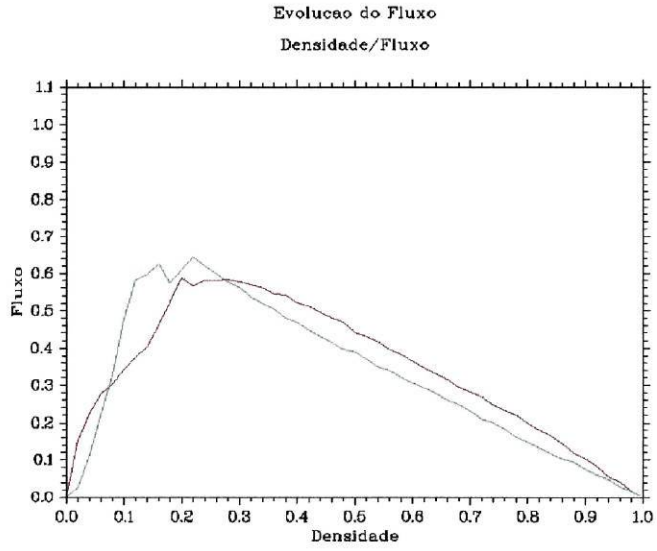
Figura~5-1: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,1 e um erro relativo de 0.1 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



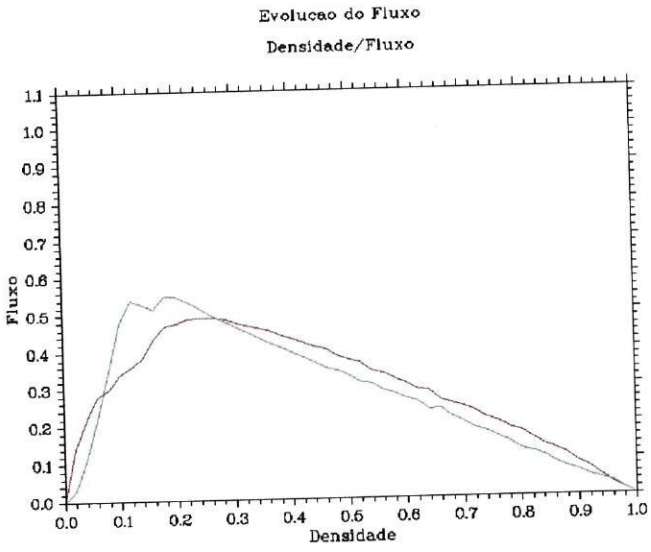
Figura~5-2: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,2 e um erro relativo de 0.1 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



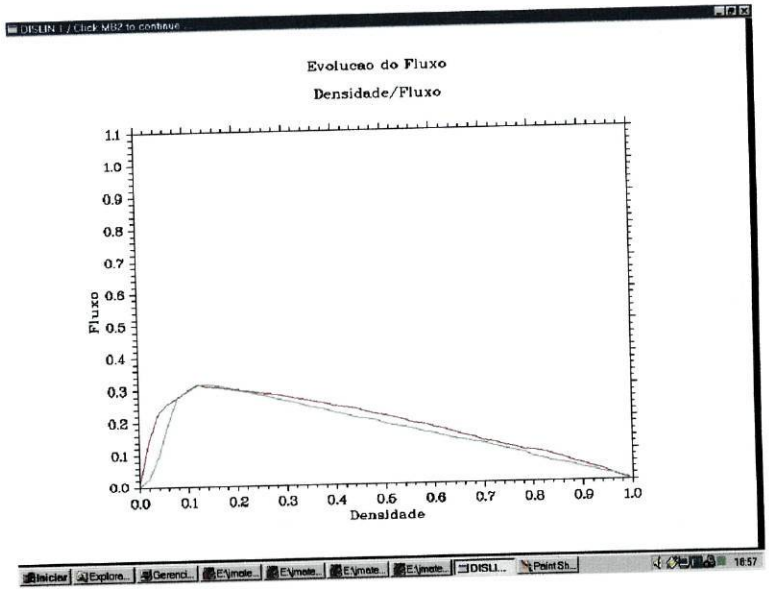
Figura~5-3: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,5 e um erro relativo de 0.1 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



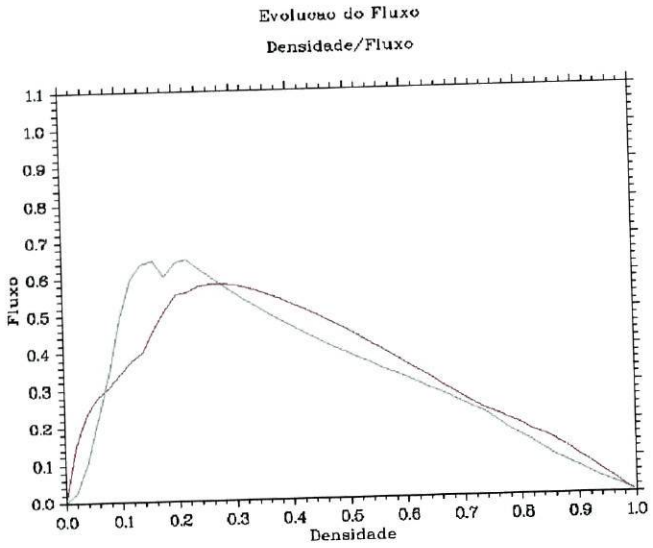
Figura~5-4: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,1 e um erro relativo de 0.01 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



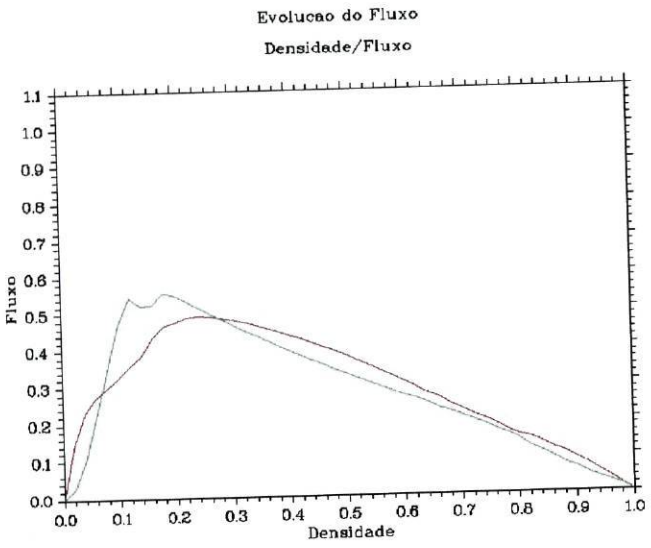
Figura~5-5: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,2 e um erro relativo de 0.01 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



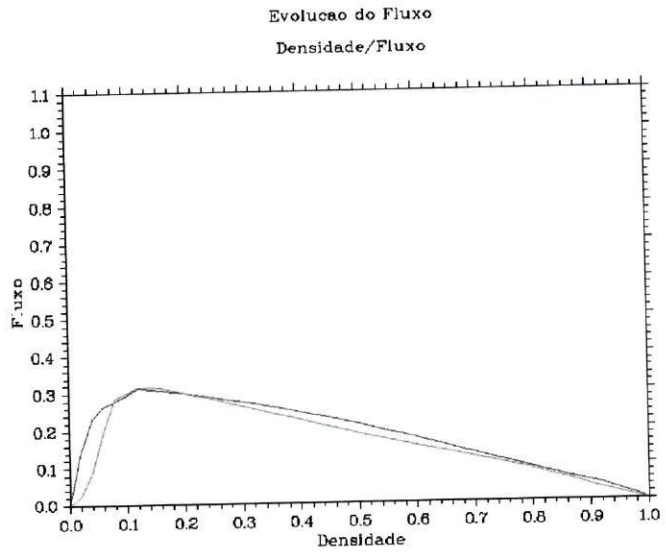
Figura~5-6: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,5 e um erro relativo de 0.01 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



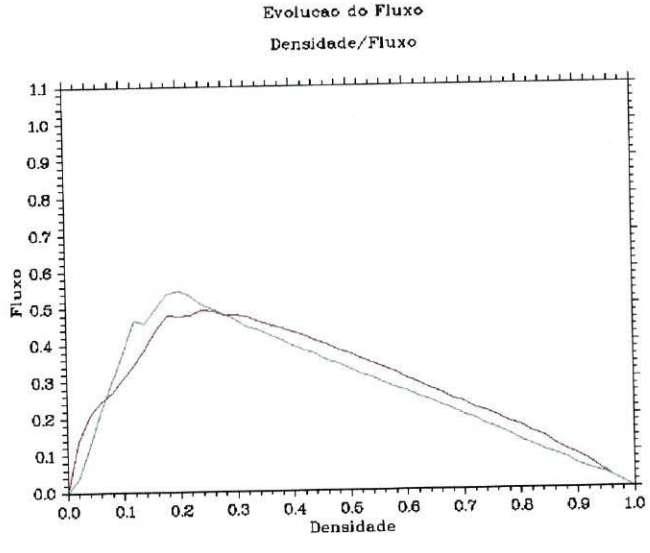
Figura~5-7: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,1 e um erro relativo de 0.001 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



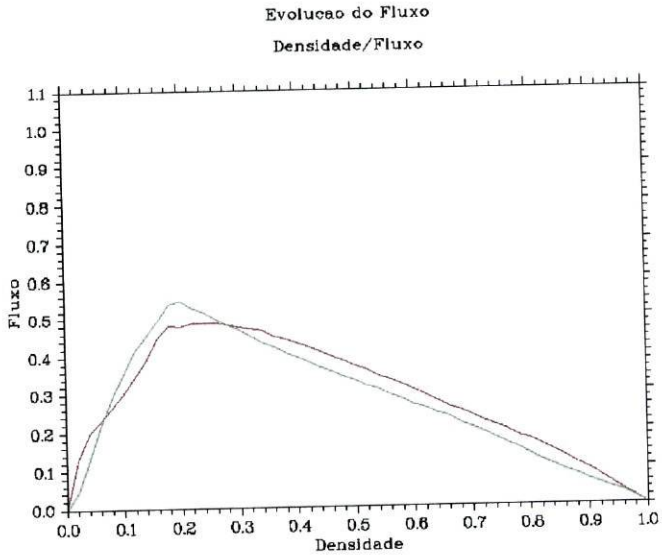
Figura~5-8: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,2 e um erro relativo de 0.001 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



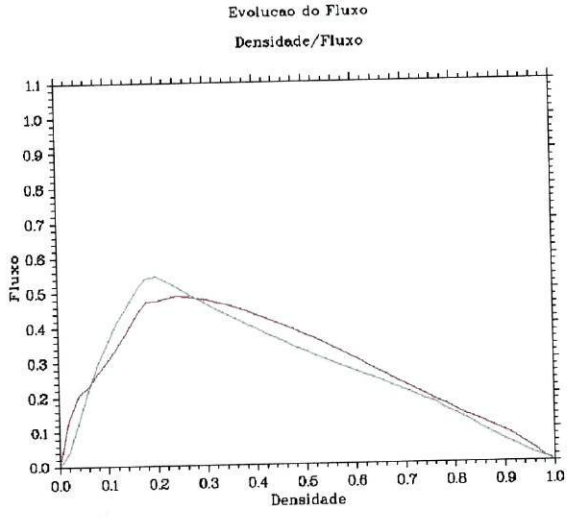
Figura~5-9: Gráfico obtido para uma estrada com 500 sítios, uma probabilidade de reduzir a velocidade de 0,5 e um erro relativo de 0.001 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



Figura~5-10: Gráfico obtido para uma estrada com 1000 sítios, uma probabilidade de reduzir a velocidade de 0,2 e um erro relativo de 0.01 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



Figura~5-11: Gráfico obtido para uma estrada com 1500 sítios, uma probabilidade de reduzir a velocidade de 0,2 e um erro relativo de 0.01 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)



Figura~5-12: Gráfico obtido para uma estrada com 1500 sítios, uma probabilidade de reduzir a velocidade de 0,2 e um erro relativo de 0.001 (Curva Vermelha-Faixa da Direita, Curva Verde-Faixa da Esquerda)

Para este modelo apresentamos o seguinte quadro, com os valores de tempo de CPU gastos na obtenção de alguns gráficos, usando um computador com processador Pentium III a 600 MHz e 128 Mbytes de memória RAM.

Dimensão da estrada (Número de Sítios)	Probabilidade de reduzir a velocidade	Erro relativo	Tempo de CPU
500	0,1	0,1	4m:54s
500	0,2	0,1	4m:59s
500	0,5	0,1	5m:04s
500	0,1	0,01	59m:35s
500	0,5	0,01	1h:08m:26s
500	0,1	0,001	63h:49m:48s
500	0,2	0,001	55h:40m:25s
500	0,5	0,001	71h:13m:57s
1000	0,2	0,01	3h:13m:31s
1500	0,2	0,001	464h:01m:52s

Capítulo 6

O Modelo para Tráfego Urbano

Neste modelo, a mudança de faixa, continua a ter o papel mais importante, referindo que um dos seguintes casos pode levar a que esta ocorra:

1- Para ganhar velocidade

Se o condutor se encontra na faixa da direita (faixa lenta) e verifica que o tráfego na outra faixa é melhor, isto é, a segunda faixa tem um fluxo mais baixo e uma maior velocidade, então com probabilidade P_muda ele mudará de faixa e com probabilidade 1-P_muda ele não mudará.

2- Para evitar obstáculos

Obstáculos, tais como veículos na entrega de correio, paragens de autocarros ou veículos que pretendem virar à direita (ou à esquerda) e por isso reduzem a sua velocidade, que aparecem ao acaso com probabilidade P_obs. Estes obstáculos estão localizados na faixa da direita com probabilidade P_obs_dir e com probabilidade 1-P_obs_dir na faixa da esquerda. Os veículos movem-se de modo a ultrapassar o obstáculo quando entram na zona obstruída.

3- Para virar no próximo cruzamento

Se o condutor que está na faixa da direita pretende virar à esquerda, ou vice-versa, uma mudança de faixa tem que ocorrer e o momento dessa mudança depende da proximidade do cruzamento.

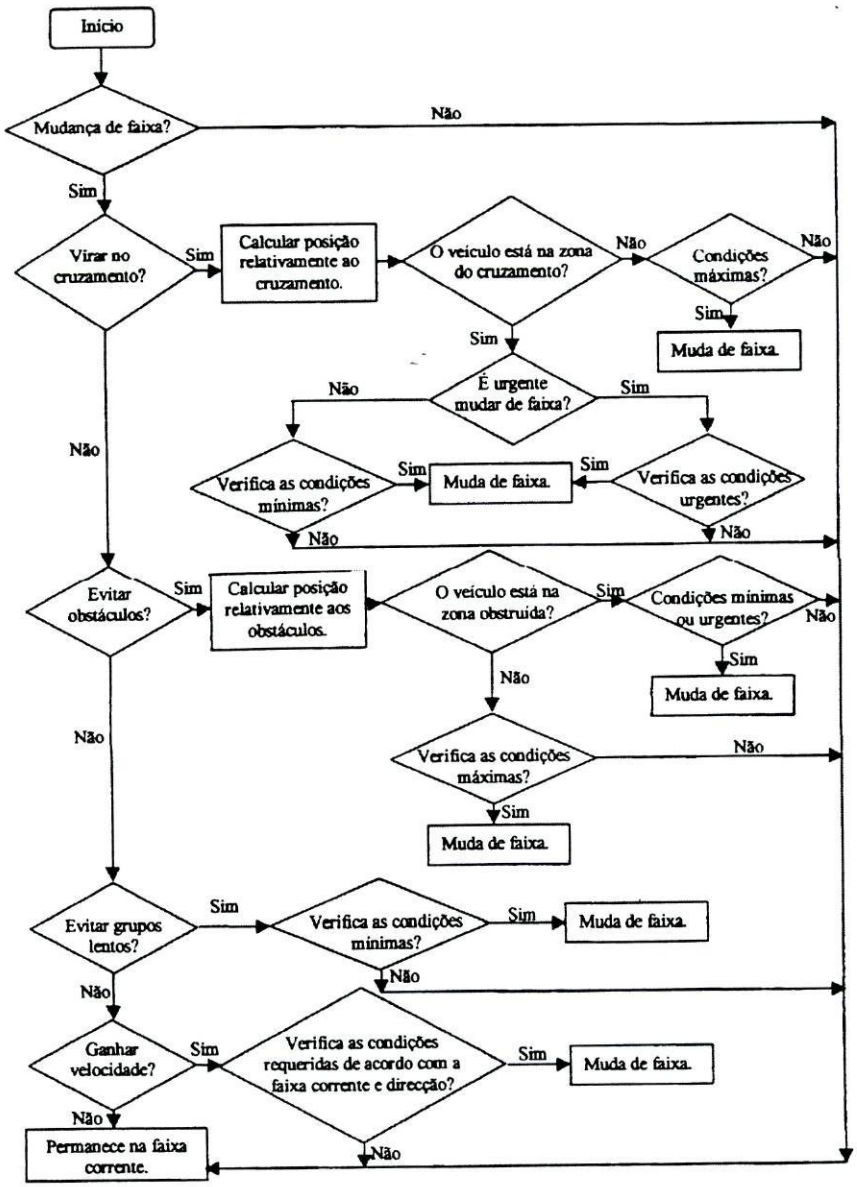
4- Para evitar grupos lentos na faixa rápida

Visto que todos consideram a faixa rápida como a melhor, podem-se formar grupos de veículos em movimento lento nessa faixa. Nesse caso, os veículos podem mudar da faixa rápida para a lenta, de forma a evitar esses grupos. A mudança pode ocorrer desde que a faixa lenta a permita.

De referir que ao contrário da primeira motivação, que se pode aplicar indiferentemente

aos dois tipos de tráfego, tanto em áreas urbanas, como fora delas, as três restantes são mais específicas para tráfego em áreas urbanas, embora possam ocorrer fora delas, mas não com tanta frequência ou relevância.

O modelo para tráfego urbano baseia-se em regras que assentam no seguinte esquema:


Início
Mudança de faixa?
Não
Sim
Virar no cruzamento?
Sim
Calcular posição relativamente ao cruzamento.
O veículo está na zona do cruzamento?
Não
Condições máximas?
Não
Sim
Muda de faixa.
Sim
É urgente mudar de faixa?
Não
Sim
Não
Verifica as condições mínimas?
Sim
Muda de faixa.
Sim
Verifica as condições urgentes?
Não
Não
Evitar obstáculos?
Sim
Calcular posição relativamente aos obstáculos.
O veículo está na zona obstruída?
Sim
Condições mínimas ou urgentes?
Não
Sim
Muda de faixa.
Não
Não
Verifica as condições máximas?
Não
Sim
Muda de faixa.
Evitar grupos lentos?
Sim
Verifica as condições mínimas?
Sim
Muda de faixa.
Não
Não
Ganhar velocidade?
Sim
Verifica as condições requeridas de acordo com a faixa corrente e direcção?
Sim
Muda de faixa.
Não
Não
Permanece na faixa corrente.

Figura~6-1:

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusões

Tendo em consideração os resultados obtidos, podemos afirmar que os modelos estudados e implementados produzem resultados satisfatórios. Os resultados obtidos sempre que comparados com outros publicados em artigos coincidem, ou melhor, são semelhantes, o que vem mostrar que a implementação é coerente com outras já existentes e embora nalguns casos se tenha efectuado adaptações próprias, podemos garantir que os modelos funcionam e a sua implementação está correcta.

A implementação pode ser adaptada a novas situações e a outros tipos de tráfego que introduzirão novas variantes nos modelos, variantes essas que usualmente se designam por regras de tráfego automóvel.

O trabalho contém uma forte componente computacional, o que por vezes o tornou bastante moroso, tanto a nível de implementação como de obtenção de resultados, apesar dos meios computacionais usados serem bastante poderosos. Como foi referido atrás para a obtenção de um gráfico houve um gasto de tempo de CPU de 464h 01m 52s, com um processador Pentium III a 600 MHz e com 128Mbytes de memória RAM, o que comprova a morosidade.

Os modelos apresentam algumas limitações e referem-se a casos particulares, embora já englobem uma grande variedade de tráfego automóvel e os tipos mais importantes na actualidade, que são o tráfego com multifaixas e dentro deste especificamente o tráfego urbano.

7.2 Trabalho Futuro

Este trabalho poderá sempre ser actualizado para novos modelos, uma vez que há código que se mantém em todos os modelos e a nível de programação há sempre a possibilidade de os

algoritmos serem optimizados.

Na prática, este trabalho poderá ser utilizado por organizações que tratem de tráfego automóvel, sendo nesse caso necessário adaptar o programa às especificidades do estudo encomendado, pois cada caso será um caso e terá as suas particularidades. Poderá ser usado para estradas já existentes, como forma de estudo das condições actuais ou para a construção de novas estradas, onde se poderão tirar conclusões àcerca das infraestruturas a construir.

Bibliografia

[1] M. Schreckenberg, A. Schadschneider, K. Nagel e N. Ito, *Discrete stochastic models for traffic flow*, Phys. Rev. E51 2339, 1995.

[2] Andreas Schadschneider, *The Nagel-Schreckenberg model revisited,* 1999.

[3] Haki Hammad, *Cellular Automata Models for Traffic Flow in Urban Networks, Ph. D.*; School of Computer Applications, Dublin City University; 1998.

[4] Debashish Chowdhury, Ludger Santen, Andreas Schadschneider, *Statistical Physics of Vehicular Traffic and Some Related Systems*, Physics Reports 329, 199; 2000

Anexo A: Código em MatLab

```
**************************************************************************

File temp6prof.m

function temp6
clear;clc;
global estrada fluxo L t % tMax
%++++++++++++++++++++++++++++ parâmetros ++++++++++++++++++++++
L=100; % numero de sitios da estrada
%# tMax=100; %tempo máximo, numero de iterações
densidade=.3; % densidade do tráfego
N=fix(densidade*L); % numero de veiculos
Vmax=5; % velocidade máxima
prob=0.5; % probabilidade de reduzir a velocidade
%++++++++++++++++++++++++++++ inicialização ++++++++++++++++++++
%! estrada(1:N) = 0; % colocar carros, nos "primeiros" N sitios da estrada
% circular, com velocidade nula
%! estrada(N+1:L) = -1; % Inicializar os restantes sitios da estrada,
% sem veiculos
estrada(1:L) = -1;
estrada(permuta(L,N))=0; % colocados parados em posições aleatórias
%++++++++++++ imagem da estrada gerada ++++++++++++++++++++++++++++
graficola
%++++++++++++++++++++++ evolução ++++++++++++++++++++++++++++++++
t=0;
fluxo=0;
variancia=0;
```

```
%# for t=1:tMax, % numero de iterações
grafico2a
while (variancia >= t*(.1*fluxo)^2) | t < 30
t=t+1;
estrada_nova(1:L)=-1; % inicialização de uma variável auxiliar, onde
% vão ser colocados os veiculos
passam=0;
i=1;
while i<=L % procura os sitios da estrada com veiculo
% e aplica as regras de actualização de velocidade
gap=1;
if estrada(i)> -1 % se o sitio tem veiculo
% calculo da distancia ao veiculo seguinte
while estrada(1+mod(i+gap-1,L))==-1,
gap=gap+1;
end
% Se a velocidade do veiculo é inferior à velocidade máxima
% então a velocidade pode ser aumentada de uma unidade
if estrada(i) < Vmax
estrada(i)=estrada(i)+1;
end
% Se a distancia ao veiculo seguinte é inferior ou igual
% à velocidade actual do veiculo, então a sua velocidade
% tem que ser reduzida e passa a ter o valor da distância
% ao veiculo seguinte menos uma unidade, para que não haja
% colisão
if gap <= estrada(i)
estrada(i)=gap-1;
end
% Com uma probabilidade "prob" e se a velocidade for superior
% a zero, a velocidade do veiculo é diminuida de uma unidade
if (rand <= prob) & (estrada(i)>0)
estrada(i) = estrada(i)-1;
```

```
end
estrada_nova(1+mod(i+estrada(i)-1,L))=estrada(i);
if i+estrada(i) > L
passam = passam + 1;
end
end % if, -actualização do valor da velocidade de cada veiculo
i = i+gap;
end % while, -todos os sitios da estrada foram actualizados
estrada = estrada_nova; % atribui à variável estrada o valor da
% variável estrada_nova, para usar a
% a variável estrada no próximo tempo
fluxo = fluxo*(1-1/t)+passam/t;
variancia = variancia*(t-2+eps)/(t-1+eps)+t*(passam-fluxo)^2/(t-1+eps)^2;
%++++++++++++ imagem da estrada gerada +++++++++++++++++++
grafico1b
%+++++++++++++++++++ gráfico fundamental +++++++++++++++++++++++
grafico2b
end % while
fluxo
variancia
t
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function P = permuta(n,r)
% n = nº de simbolos inicial
% r = nº de elementos do subconjunto a extrair
V = (1:n);
for k=n:-1:1+n-r
u = 1 + fix(k*rand);
troca = V(u);
V(u) = V(k);
V(k) = troca;
end
P = V(n-r+1:n);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function grafico1a
global estrada L % tMax
figure(1)
clf
hold on
axis([1 L -100 0])
% axis off
for i=1:L
if estrada(i)>-1
plot(i,0,'k.');
end
end
% pause(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function grafico1b
global estrada L t
figure(1)
for i=1:L
if estrada(i)>-1
plot(i,-t,'k.');
end
end
% pause(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function grafico2a
figure(2)
clf
hold on
axis([1 100 0 1])
% axis off
xlabel('tempo')
ylabel('fluxo')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function grafico2b
global fluxo t
figure(2)
plot(t,fluxo,'k.')
% pause(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Anexo B: Código em C para Modelo de 1 Faixa

```
*****************************************************************************

File dens_fluxo.c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "dislin.h"
/* Constante suficientemente pequena */
#define eps 1.0E-32
/* Variaveis Globais*/
int *estrada;
float fluxo;
float densidade;
int L; /* numero de sitios na estrada */
int N; /* numero de veiculos */
int Vmax; /* velocidade maxima */
float prob; /* probabilidade de reduzir a velocidade */
/* variaveis auxiliares */
int *V;
/* ROTINAS */
/* Função que calcula o minimo de dois números inteiros */
int min(x,y)
int x,y;
```

```
{
return(x<y?x:y);
}
/* Função que calcula o máximo de dois números inteiros */
int max(x,y)
int x,y;
{
return(x>y?x:y);
}
/* Função que calcula o quadrado de um número real */
float sqr(x)
float x;
{
return(x*x);
}
/* Rotina auxiliar que de n elementos iniciais vai extrair r, aleatoriamente */
void permuta(n,r)
int n,r;
{
int k,u,troca;
V = (int *) malloc((n+1)*sizeof(int));
if ( V==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(1);
}
for( k=1 ; k<=n ; k++ )
V[k] = k;
for( k=n ; k>=n-r+1 ; k- )
{
u = 1+ (int) k*(float) rand()/RAND_MAX;
troca = V[u];
V[u] = V[k];
```

```
V[k] = troca;
}
}
/* Inicializa os parametros (Variáveis) */
void inicializa_param()
{
L = 10000;
N = ((int) (densidade*L));
Vmax = 5;
prob = 0.5;
}
/* Inicializa a "estrada" e dá-lhe uma configuração inicial aleatória */
/* Distribui os veiculos em sitios aleatórios, com velocidade aleatória */
void inicializacao()
{
int i;
estrada = (int *) malloc((L+1)*sizeof(int));
if ( estrada==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(1);
}
/* Atribui a todos os sitios o valor -1, ou seja, sem veiculos */
for( i=1 ; i<=L ; i++ )
estrada[i] = -1;
/* Dos L sitios da estrada, vai extrair N, que vao ser os que vão conter veiculos */
permuta(L,N);
/* Aos N sitios calculados, atribui-lhe um veiculo, com velocidade entre 0 (parado)
e a velocidade máxima */
for( i=1 ; i<=N ; i++ )
estrada[V[L-N+i]] = (int) ((Vmax+1)*(((float) rand())/RAND_MAX));
}
/* Rotina responsável pela actualizacao da posicao dos veiculos em cada iteracao */
```

```
int evolui()
{
int *estrada_nova;
int i, gap, estrada1;
int passam;
estrada_nova = (int *) malloc((L+1)*sizeof(int));
if ( estrada_nova==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(1);
}
/* Inicialização de uma variável auxiliar, onde vão ser colocados os veículos */
for( i=1 ; i<=L ; i++ )
estrada_nova[i] = -1;
i = 1;
/* Determina o 1º veiculo */
while(estrada[1+(i-1) % L] == -1)
i += 1;
passam = 0;
while (i <= L) /* Procura os sitio da estrada com veiculo e aplica as regras de
actualização da velocidade associada a cada veiculo */
{
estrada1 = estrada[i];
gap = 1;
/* Calcula a distancia ao veiculo seguinte */
while(estrada[1+(i+gap-1) % L] == -1)
gap += 1;
/* Se a velocidade do veiculo é inferior à velocidade máxima, então a velocidade
que lhe está associada pode ser aumentada de uma unidade */
estrada1 = min(estrada1+1,Vmax);
/* Se a distancia ao veiculo seguinte é inferior ou igual à velocidade actual
do veiculo, então a sua velocidade tem que ser reduzida e passa a ter o valor
da distancia ao veiculo seguinte menos uma unidade, para que dessa forma não
```

```
haja colisão */
estrada1 = min(gap-1,estrada1);
/* Com uma probabilidade definida "prob" e se a velocidade for superior a zero,
a velocidade do veiculo é diminuida de uma unidade */
if (((float)rand()/RAND_MAX) <= prob)
estrada1 = max(estrada1-1,0);
estrada_nova[1+(i+estrada1-1) % L] = estrada1;
/* Se há a passagem de um veiculo no local considerado o inicio da estrada,
a variável passam toma o valor 1, para o cálculo do fluxo */
if (i+estrada1 > L)
passam = 1;
i = i+gap;
}
/* Atribui à variável estrada o valor da variável estrada_nova, para usar a
variável estrada (actualizada) na próxima iteração */
for( i=1 ; i<=L ; i++ )
estrada[i] = estrada_nova[i];
free(estrada_nova);
return(passam);
}
float calcula_fluxo()
{
int i,b;
float fluxoT, varianciaT;
/* Chamada das funções que vão efectuar a inicialização das variáveis */
inicializa_param ();
inicializacao ();
/* Se o numero de veiculos é zero ou os sitios estão todos ocupados o fluxo é zero,
caso contrário vai calcular o fluxo */
if ((N==0) || (N==L))
fluxoT=0;
else
{
```

```
/* Faz um "aquecimento inicial" com L iterações, sem interessar o valor do
fluxo obtido */
for( i=1 ; i<=L ; i++ )
evolui();
b = 0;
fluxoT = 0;
varianciaT = 0;
/* gera valores para o fluxo até obter uma boa estatistica, efectuando no minimo
30 iterações para garantir bons valores */
while ((varianciaT >= (b*sqr(0.01*fluxoT))) || (b < 30) )
{
b += 1;
fluxo = 0;
/* Faz L iterações para o cálculo do fluxo */
for( i=1 ; i<=L ; i++ )
fluxo = fluxo*(1-(float)1/i)+(float)evolui()/i;
fluxoT = fluxoT*(1-(float)1/b)+fluxo/b;
varianciaT = varianciaT*(b-2)/(b-1+eps)+b*(sqr((fluxo-fluxoT)/(b-1+eps)));
}
}
free(estrada);
free(V);
return fluxoT;
}
main()
{
int i;
float xray[100], yray[100];
/* Determina uma semente que vai ser usada pela função rand, que gera
numeros aleatórios */
srand (time(NULL));
/* Para vários valores de densidade vai determinar o correspondente fluxo */
for (i = 0; i <100 ; i++)
```

```
{
densidade= (float)(i+1)/100;
printf ("%s%f\n","densidade=",densidade);
xray[i] = densidade ;
yray[i] = calcula_fluxo ();
}
/******** instrucoes graficas *************/
metafl("cons"); /*define o ficheiro metafile */
disini();
pagera();
complx();
axspos(450,1800);
axslen(2200,1200);
name("Densidade","x");
name("Fluxo","y");
ticks(5,"xy");
titlin("Evolucao do Fluxo",1);
titlin("Densidade/Fluxo",3);
graf(0.0,1.0,0.0,0.1,0.0,0.5,0.0,0.1);
title();
color("white");
curve(xray,yray,100);
color("fore");
dash();
xaxgit();
disfin();
return (0);
}
```

****

File Ite_fluxo.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>
#include <time.h>
#include "dislin.h"
/* Constante suficientemente pequena */
#define eps 1.0E-32
/* Variaveis Globais*/
int *estrada;
float fluxo;
float densidade;
int L; /* numero de sitios na estrada */
int N; /* numero de veiculos */
int Vmax; /* velocidade maxima */
float prob; /* probabilidade de reduzir a velocidade */
/* variaveis auxiliares */
int *V;
/* ROTINAS */
/* Função que calcula o minimo de dois números inteiros */
int min(x,y)
int x,y;
{
return(x<y?x:y);
}
/* Função que calcula o máximo de dois números inteiros */
int max(x,y)
int x,y;
{
return(x>y?x:y);
}
/* Função que calcula o quadrado de um número real */
float sqr(x)
float x;
{
return(x*x);
```

```
}
/* Rotina auxiliar que de n elementos iniciais vai extrair r, aleatoriamente */
void permuta(n,r)
int n,r;
{
int k,u,troca;
V = (int *) malloc((n+1)*sizeof(int));
if ( V==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(1);
}
for( k=1 ; k<=n ; k++ )
V[k] = k;
for( k=n ; k>=n-r+1 ; k- )
{
u = 1+ (int) k*(float) rand()/RAND_MAX;
troca = V[u];
V[u] = V[k];
V[k] = troca;
}
}
/* Inicializa os parametros (Variáveis) */
void inicializa_param()
{
L = 10000;
densidade = 0.5;
N = ((int) (densidade*L));
Vmax = 5;
prob = 0.5;
}
/* Inicializa a "estrada" e dá-lhe uma configuração inicial aleatória */
/* Distribui os veiculos em sitios aleatórios, com velocidade aleatória */
```

```
void inicializacao()
{
int i;
estrada = (int *) malloc((L+1)*sizeof(int));
if ( estrada==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(1);
}
/* Atribui a todos os sitios o valor -1, ou seja, sem veiculos */
for( i=1 ; i<=L ; i++ )
estrada[i] = -1;
/* Dos L sitios da estrada, vai extrair N, que vao ser os que vão conter veiculos */
permuta(L,N);
/* Aos N sitios calculados, atribui-lhe um veiculo, com velocidade entre 0 (parado) e a
velocidade máxima */
for( i=1 ; i<=N ; i++ )
estrada[V[L-N+i]] = (int) ((Vmax+1)*(((float) rand())/RAND_MAX));
}
/* Rotina responsável pela actualizacao da posicao dos veiculos em cada iteracao */
int evolui()
{
int *estrada_nova;
int i, gap, estrada1;
int passam;
estrada_nova = (int *) malloc((L+1)*sizeof(int));
if ( estrada_nova==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(1);
}
/* Inicialização de uma variável auxiliar, onde vão ser colocados os veículos */
for( i=1 ; i<=L ; i++ )
```

```
estrada_nova[i] = -1;
i = 1;
/* Determina o 1º veiculo */
while(estrada[1+(i-1) % L] == -1)
i += 1;
passam = 0;
while (i <= L) /* Procura os sitio da estrada com veiculo e aplica as regras de actualização
da velocidade
associada a cada veiculo */
{
estrada1 = estrada[i];
gap = 1;
/* Calcula a distancia ao veiculo seguinte */
while(estrada[1+(i+gap-1) % L] == -1)
gap += 1;
/* Se a velocidade do veiculo é inferior à velocidade máxima, então a velocidade que lhe
está associada
pode ser aumentada de uma unidade */
estrada1 = min(estrada1+1,Vmax);
/* Se a distancia ao veiculo seguinte é inferior ou igual à velocidade actual do veiculo,
então a sua velocidade tem que ser reduzida e passa a ter o valor da distancia ao
veiculo seguinte menos uma unidade, para que dessa forma não haja colisão */
estrada1 = min(gap-1,estrada1);
/* Com uma probabilidade definida "prob" e se a velocidade for superior a zero,
a velocidade do veiculo é diminuida de uma unidade */
if (((float)rand()/RAND_MAX) <= prob)
estrada1 = max(estrada1-1,0);
estrada_nova[1+(i+estrada1-1) % L] = estrada1;
/* Se há a passagem de um veiculo no local considerado o inicio da estrada,
a variável passam toma o valor 1, para o cálculo do fluxo */
if (i+estrada1 > L)
passam = 1;
i = i+gap;
```

```
}
/* Atribui à variável estrada o valor da variável estrada_nova, para usar a variável estrada
(actualizada) na próxima iteração */
for( i=1 ; i<=L ; i++ )
estrada[i] = estrada_nova[i];
free(estrada_nova);
return(passam);
}
main()
{
int i,b;
float fluxoT, varianciaT;
float xray[30], yray[30];
inicializa_param ();
inicializacao ();
/* Determina uma semente que vai ser usada pela função rand, que gera numeros aleatórios
*/
srand (time(NULL));
/* Faz um "aquecimento inicial" com L iterações, sem interessar o valor do fluxo obtido */
for( i=1 ; i<=L ; i++ )
evolui();
b = 0;
fluxoT = 0;
varianciaT = 0;
/* gera valores para o fluxo até obter uma boa estatistica, efectuando no minimo
30 iterações para garantir bons valores */
while ((varianciaT >= (b*sqr(0.01*fluxoT))) || (b < 30) )
{
b += 1;
fluxo = 0;
/* Faz L iterações para o cálculo do fluxo */
for( i=1 ; i<=L ; i++ )
fluxo = fluxo*(1-(float)1/i)+(float)evolui()/i;
```

```
fluxoT = fluxoT*(1-(float)1/b)+fluxo/b;
varianciaT = varianciaT*(b-2)/(b-1+eps)+b*(sqr((fluxo-fluxoT)/(b-1+eps)));
yray[b-1] = fluxoT;
printf ("%s%f\n","fluxo=",fluxoT);
printf ("%s%d\n","iteracoes=",b);
}
free(estrada);
free(V);
/******** instrucoes graficas **********/
for (i = 0; i <30 ; i++)
xray[i] = i+1 ;
metafl("cons"); /*define o ficheiro metafile */
disini();
pagera();
complx();
axspos(450,1800);
axslen(2200,1200);
name("Iteracoes","x");
name("Fluxo","y");
labdig(-1,"x");
ticks(5,"xy");
titlin("Evolucao do Fluxo",1);
titlin("Iteracoes/Fluxo",3);
graf(0.,30.,0.,5.,0.0,0.5,0.0,0.1);
title();
color("red");
curve(xray,yray,30);
color("fore");
dash();
xaxgit();
disfin();
return (0);
}
```

Anexo C: Código em C para Modelo de 2 Faixas

```
*************************************************************************

  File CALCULAFLUXO.C

  #include <stdlib.h>
  #include <stdio.h>
  double evolui(char*, char*, unsigned int);
  void inicializa(double);
  void muda(void);
  extern unsigned int L, N, NDir, NEsq;
  extern char *pEstradaDir, *pEstradaEsq, *pEstradaDirAux, *pEstradaEsqAux;
  extern FILE *pFileOut;
  extern double ERRO;
  #define sqr(x) ((x)*(x)) /* Função que calcula o quadrado de um número real */
  void CalculaFluxo(double densidade, double *FDir, double *FEsq)
  {
  double
  fluxoDir=0., fluxoEsq=0.,
  //fluxoTotal=0.,
  b=1., i=0.,
  fluxoTotalDir=0., fluxoTotalEsq=0.,
  varianciaDir=0., varianciaEsq=0.;
  /* Chamada das funções que vão efectuar a inicialização das variáveis */
  inicializa(densidade);
  /* Se o numero de veiculos é zero ou os sitios estão todos ocupados, em
```

```
ambas as faixas, o fluxo é zero, caso contrário vai calcular o fluxo */
if ((N !=0) && (N != 2*L))
{
/* Faz um "aquecimento inicial" com L iterações */
for( i=0. ; i<L ; i++ )
{
muda();
evolui(pEstradaEsq,pEstradaEsqAux, NEsq);
evolui(pEstradaDir,pEstradaDirAux,NDir);
}
/* Faz uma primeira amostragem */
for( i=1. ; i<(L+1) ; i++ )
{
muda();
fluxoTotalEsq = fluxoTotalEsq*(1.-1./i)+evolui(pEstradaEsq,pEstradaEsqAux,NEsq)/i;
fluxoTotalDir = fluxoTotalDir*(1.-1./i)+evolui(pEstradaDir,pEstradaDirAux,NDir)/i;
}
/* gera valores para o fluxo até obter uma boa estatistica, efectuando no minimo
30 iterações para garantir o teorema do limite central */
while (
(b < 30.)
||
(varianciaDir > (b*sqr(ERRO*fluxoTotalDir)))
||
(varianciaEsq > (b*sqr(ERRO*fluxoTotalEsq)))
)
{
b++;
fluxoDir = 0.;
fluxoEsq = 0.;
/* Faz L iterações para o cálculo do fluxo */
for( i=1. ; i<(L+1) ; i++ )
{
```

```
muda();
fluxoEsq = fluxoEsq*(1.-1./i)+evolui(pEstradaEsq,pEstradaEsqAux,NEsq)/i;
fluxoDir = fluxoDir*(1.-1./i)+evolui(pEstradaDir,pEstradaDirAux,NDir)/i;
}
fluxoTotalDir = fluxoTotalDir*(1.-1./b)+fluxoDir/b;
varianciaDir = varianciaDir*(b-2.)/(b-1.)+b*(sqr((fluxoDir-fluxoTotalDir)/(b-1.)));
fluxoTotalEsq = fluxoTotalEsq*(1.-1./b)+fluxoEsq/b;
varianciaEsq = varianciaEsq*(b-2.)/(b-1.)+b*(sqr((fluxoEsq-fluxoTotalEsq)/(b-1.)));
}
}
//fluxoTotal = fluxoTotalDir + fluxoTotalEsq;
*FEsq = fluxoTotalEsq;
*FDir = fluxoTotalDir;
//printf("%s%f\n","fluxo total=", fluxoTotal);
//fprintf(pFileOut, "%s\t%f\n","fluxo total=",fluxoTotal);
printf("%s%f\t%s%f\n","fluxo dir = ", fluxoTotalDir,"fluxo esq = ",fluxoTotalEsq);
fprintf(pFileOut,"%s%f\t%s%f\n","fluxo dir = ", fluxoTotalDir,"fluxo esq = ",fluxoTotalEsq);
//return fluxoTotal;
}
```

****

File Corrente.c

```
/*////////////////////////////////////////////////////
Tráfego em auto-estrada com 2 faixas
////////////////////////////////////////////////////*/
#include "corrente.h"
////////////////////////////////////////////////////////////////////////////////
int main(void)
{
unsigned int i=0;
double densidade=0., FDir, FEsq;
float xray[101], yDray[101], yEray[101];
janela(); // gera janela para escolha das características do sistema
```

```
    alocar(); //reserva memória para os arrays compridos para não ter que repetir
    printf("Tamanho= %d\tProb.= %.1f\tVmax= %d\t\tErro= %.3f\n\n",L,prob,Vmax,ERRO);
// escreve cabeçalhos
    fprintf(pFileOut,"Tamanho= %d\tProb.= %.1f\tVmax= %d\n\n",L,prob,Vmax);
    // Para vários valores de densidade vai determinar o correspondente fluxo
    for (i = 0; i <51 ; i++)
    {
    densidade= i/50.;
    printf ("%s%.2f\t","densidade = ",densidade);
    fprintf(pFileOut, "%s%.2f\t","densidade = ",densidade);
    xray[i] = densidade ;
    CalculaFluxo(densidade, &FDir, &FEsq);
    yDray[i] = FDir;
    yEray[i] = FEsq;
    }
    grafico(xray,yDray,yEray);
    return (0);
    }
    ********************************************************************************

    File Corrente.h

    #include <stdio.h>
    #include <stdlib.h>
    #include <math.h>
    #include <time.h>
    /* Variaveis Globais*/
    char
    *pEstradaDir=NULL,
    *pEstradaEsq=NULL,
    *pEstradaDirAux=NULL,
    *pEstradaEsqAux=NULL;
    unsigned int
    NDir, NEsq, /* numero de veiculos na faixa da direita e na da esquerda, respectivamente
*/
```

```
L, /* numero de sitios na estrada */
N; /* numero de veiculos */
unsigned char Vmax; /* velocidade maxima */
double prob, /* probabilidade de reduzir a velocidade */
ERRO; // erro relativo
FILE *pFileOut=NULL; // ponteiro para ficheiro de saida de dados
// protótipos:
void alocar(void);
void CalculaFluxo(double,double*,double*);
void grafico(float*,float*,float*);
void janela(void);
*****************************************************************************
```

File CORRENTE_ALOCAR.C

```
#include <stdlib.h>
#include <stdio.h>
extern FILE *pFileOut;
extern char
*pEstradaDir,
*pEstradaEsq,
*pEstradaDirAux,
*pEstradaEsqAux;
extern unsigned int L;
void alocar(void)
{
//.....abre ficheiro de saida de dados.....
if((pFileOut = fopen("corrente.dat", "w")) == NULL)
{
printf("can't open file\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaDir.....
pEstradaDir = malloc(L*sizeof(char));
```

```
if ( pEstradaDir==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaEsq.....
pEstradaEsq = malloc(L*sizeof(char));
if ( pEstradaEsq==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaDirAux.....
pEstradaDirAux = malloc(L*sizeof(char));
if ( pEstradaDir==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaEsqAux.....
pEstradaEsqAux = malloc(L*sizeof(char));
if ( pEstradaEsq==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
}
*****************************************************************************
```

File CORRENTE_DISLIN.C

```
#include <stdio.h>
#include <stdlib.h>
#include <dislin.h>
```

```
void janela(void);
void FunL(int);
void FunProb(int);
void FunErro(int);
void FunVmax(int);
static int
idMainLLista,
idMainVmaxLista,
idMainProbLista,
idMainErroLista;
static char
listaL[]="10|100|500|1000|1500",
listaVmax[]="1|2|5|6|10",
listaProb[]=".1|.2|.5",
listaErro[]=".1|.01|.001";
extern unsigned int L;
extern unsigned char Vmax;
extern double prob, ERRO;
void grafico(float* x,float* y1, float* y2)
{
metafl("cons"); /*define o ficheiro metafile */
page(2970,2100);
disini();
pagfll(-1);
pagera();
complx();
color ("black");
//axspos(450,1800);
//axslen(2100,1200);
name("Densidade","x");
name("Fluxo","y");
ticks(5,"xy");
titlin("Evolucao do Fluxo",1);
```

```
titlin("Densidade/Fluxo",3);
graf(0.0,1.0,0.0,0.1,0.0,1.1,0.0,0.1);
title();
color("red");
//incmrk(3);
curve(x,y1,51);
color("green");
//incmrk(4);
curve(x,y2,51);
//dash();
xaxgit();
/* Guarda o gráfico num ficheiro */
/* rtiff ("exemplo.tif");*/
color("fore");
disfin();
}
////////////////////////////////////////////////////////////////////////////////
void janela(void)
{
int idMain,
idMainL,
idMainProb,
idMainErro,
idMainVmax,
idMainLTxt,
idMainVmaxTxt,
idMainProbTxt,
idMainErroTxt;
swgtit("Escolha do sistema");
swgpop("NOOK");
swgpop("nohelp");
swgpop("noquit");
idMain=wgini("VERT"); // cria a janela principal
```

```
idMainL=wgbas(idMain,"HORI");
idMainLTxt=wglab(idMainL,"Tamanho:");
idMainLLista=wgdlis(idMainL, listaL, 2);
FunL(idMainLLista); // para o valor por defeito
swgcbk(idMainLLista,FunL);
idMainVmax=wgbas(idMain,"HORI");
idMainVmaxTxt=wglab(idMainVmax,"Vmax:");
idMainVmaxLista=wgdlis(idMainVmax, listaVmax, 2);
FunVmax(idMainVmaxLista); // para o valor por defeito
swgcbk(idMainVmaxLista,FunVmax);
idMainProb=wgbas(idMain,"HORI");
idMainProbTxt=wglab(idMainProb,"Probabilidade:");
idMainProbLista=wgdlis(idMainProb, listaProb,1);
FunProb(idMainProbLista); // para o valor por defeito
swgcbk(idMainProbLista,FunProb);
idMainErro=wgbas(idMain,"HORI");
idMainErroTxt=wglab(idMainErro,"Erro relativo:");
idMainErroLista=wgdlis(idMainErro, listaErro,2);
FunErro(idMainErroLista); // para o valor por defeito
swgcbk(idMainErroLista,FunErro);
wgok(idMain);
wgfin();
}
////////////////////////////////////////////////////////////////////////////////
void FunL(int id)
{
L=atoi(itmstr(listaL,gwglis(idMainLLista)));
}
void FunProb(int id)
{
prob=atof(itmstr(listaProb,gwglis(idMainProbLista)));
}
void FunVmax(int id)
```

```
{
Vmax=atoi(itmstr(listaVmax,gwglis(idMainVmaxLista)));
}
void FunErro(int id)
{
ERRO=atof(itmstr(listaErro,gwglis(idMainErroLista)));
}
```

****

File Evolui.c

```
/* Rotina responsável pela actualizacao da posicao dos veiculos em cada iteracao */
#include <stdlib.h>
extern unsigned int L;
extern unsigned char Vmax;
extern double prob;
#define min(x,y) (((x)<(y))?(x):(y)) // Função que calcula o minimo de dois números
inteiros
#define max(x,y) (((x)>(y))?(x):(y)) // Função que calcula o máximo
static double TRUE=1.0, FALSE=0.0;
double evolui(char *pEstrada,char *pEstradaAux, unsigned int N)
{
int i, gap, passam=FALSE;
char estrada1;
if((N<1) || ((N+1)>L))
return passam;
// Determina o 1º veiculo
i=0;
while((i<L) && (pEstrada[i] == -1))
i++;
while (i < L) /* Procura os sitio da estrada com veiculo e aplica as regras de
actualização da velocidade associada a cada veiculo */
{
estrada1 = pEstrada[i];
```

```
gap = 1;
/* Calcula a distancia ao veiculo seguinte */
while(pEstrada[(i+gap) % L] == -1)
gap++;
/* Se a velocidade do veiculo é inferior à velocidade máxima, então a velocidade
que lhe está associada pode ser aumentada de uma unidade */
estrada1 = min(estrada1+1,Vmax);
/* Se a distancia ao veiculo seguinte é inferior ou igual à velocidade actual do veiculo,
então a sua velocidade tem que ser reduzida e passa a ter o valor da distancia ao
veiculo seguinte menos uma unidade, para que dessa forma não haja colisão */
estrada1 = min(gap-1,estrada1);
/* Com uma probabilidade definida "prob" e se a velocidade for superior a zero,
a velocidade do veiculo é diminuida de uma unidade */
if (((double)rand()/(RAND_MAX+1)) < prob)
estrada1 = max(estrada1-1,0);
pEstradaAux[i]=-1;
pEstradaAux[(i+estrada1) % L]=estrada1;
/* Se há a passagem de um veiculo no local considerado o inicio da estrada,
a variável passam toma o valor 1, para o cálculo do fluxo */
if (i+estrada1 > L-1)
passam = TRUE;
i = i+gap;
}
for (i=0; i<L; i++)
pEstrada[i]=pEstradaAux[i];
return(passam);
}
*****************************************************************************
```

File Inicializa.c

```
/* Inicializa a "estrada" e dá-lhe uma configuração inicial aleatória */
/* Distribui os veiculos em sitios aleatórios, com velocidade aleatória */
#include <stdlib.h>
```

```
extern char *pEstradaDir, *pEstradaEsq, *pEstradaEsqAux, *pEstradaDirAux;
extern unsigned int NDir, NEsq, L, N;
extern unsigned char Vmax;
unsigned int NDirAux, NEsqAux;
#define min(x,y) (((x)<(y))?(x):(y)) // Função que calcula o minimo de
void permuta(char*,unsigned int);
void IniciaParam(double);
void inicializa(double densidade)
{
int i;
IniciaParam(densidade);
//.....
/* Determina quantos sitios em cada faixa que vão estar ocupados */
NDirAux=NDir = N/2;
NEsqAux=NEsq = N - NDir;
//.....
/* Distribuição dos veiculos pela faixa da direita */
/* Atribui a todos os sitios o valor -1, ou seja, sem veiculos */
for( i=0 ; i<L-NDir ; i++ )
pEstradaDir[i] = -1;
/* Aos NDir sitios calculados, atribui-lhe um veiculo, com velocidade entre 0 (parado) e a
velocidade máxima */
for( i=L-NDir ; i<L ; i++ )
pEstradaDir[i] = ((Vmax+1)*(((double) rand())/(RAND_MAX+1)));
/* Dos L sitios da faixa da direita, vai extrair NDir, que vao ser os que vão conter veiculos
*/
permuta(pEstradaDir,NDir);
//.....
/* Distribuição dos veiculos pela faixa da esquerda */
/* Atribui a todos os sitios o valor -1, ou seja, sem veiculos */
for( i=0 ; i<L-NEsq ; i++ )
pEstradaEsq[i] = -1;
```

```
    /* Aos NEsq sitios calculados, atribui-lhe um veiculo, com velocidade entre 0 (parado) e a
velocidade máxima */
    for( i=L-NEsq ; i<L ; i++ )
    pEstradaEsq[i] = ((Vmax+1)*(((double) rand())/(RAND_MAX+1)));
    /* Dos L sitios da faixa da esquerda, vai extrair NEsq, que vao ser os que vão conter veiculos
*/
    permuta(pEstradaEsq,NEsq);
    for (i=0; i<L; i++)
    {
    pEstradaEsqAux[i]=pEstradaEsq[i];
    pEstradaDirAux[i]=pEstradaDir[i];
    }
    }
    ************************************************************************************
```

File ROTINASPEQUENAS.C

```
    #include <stdlib.h>
    extern unsigned int L, N;
    void permuta(char *V, unsigned int r) // de L elementos iniciais vai extrair r, aleatoriamente
    {
    int k, u;
    char troca;
    for( k=L-1 ; k>L-r-1 ; k--)
    {
    u = k*(double) rand()/(RAND_MAX+1);
    troca = V[u];
    V[u] = V[k];
    V[k] = troca;
    }
    }
    ////////////////////////////////////////////////////////////////////////////////////
    void IniciaParam(double densidade) // Inicializa os parametros (Variáveis)
    {
```

```
N = densidade*2*L;
}
*****************************************************************************
```

File muda.c

```
/*Rotina responsável pela mudança de faixa*/
#include <stdlib.h>
extern unsigned int NDir, NEsq, L, NDirAux, NEsqAux;
extern char *pEstradaDir, *pEstradaEsq, Vmax,
*pEstradaDirAux, *pEstradaEsqAux;
char Voff=8;
double probR=.05;
void muda(void)
{
unsigned int i,
dist_o_b=0, //distancia ao veiculo anterior na outra faixa
dist, //distancia ao veiculo seguinte na mesma faixa
dist_o; //distancia ao veiculo seguinte na outra faixa
char bool;
/* Se o numero de veiculos na faixa da direita é diferente de zero, aplica as
regras de mudanca de faixa DA DIREITA PARA A ESQUERDA */
if (NDir > 0)
{
i = 0;
/* Determina o 1º veiculo na faixa da direita*/
while(pEstradaDir[i] == -1)
i++;
while (i < L) /* Procura os sitios da estrada com veiculo e aplica as regras de
mudança de faixa DA DIREITA PARA A ESQUERDA */
{
dist_o_b=0;
/* Calcula a distancia ao veiculo anterior na outra faixa */
while ((pEstradaEsq[(L+i-dist_o_b) % L] == -1) && (dist_o_b<L))
```

```
dist_o_b++;
dist = 1;
/* Calcula a distancia ao veiculo seguinte na mesma faixa */
while (pEstradaDir[(i+dist) % L] == -1)
dist++;
dist_o = 1;
/* Calcula a distancia ao veiculo seguinte na outra faixa */
while ((pEstradaEsq[(i+dist_o) % L] == -1) && (dist_o<L))
dist_o++;
/* Verifica as condicoes de mudança da faixa DA DIREITA PARA A ESQUERDA */
if (
(Vmax < dist_o_b) // manter dist. de segur. ao anterior na outra faixa
&&
((Vmax +1)> dist) // nesta faixa não posso ir tão rápido como desejo
&&
((dist_o +1)> dist) // a outra faixa não é pior
)
{
pEstradaEsqAux[i] = pEstradaDir[i];
NEsqAux++;
pEstradaDirAux[i]=-1;
NDirAux--;
}
i += dist; /* Avança para o próximo veiculo */
}
}
/* Se o numero de veiculos na faixa da esquerda é diferente de zero, aplica as
regras de mudanca de faixa DA ESQUERDA PARA A DIREITA */
if (NEsq > 0 )
{
i = 0;
/* Determina o 1º veiculo na faixa da esquerda*/
while(pEstradaEsq[i] == -1)
```

```
i++;
while (i < L) /* Procura os sitios da estrada com veiculo e aplica as regras de
mudança de faixa DA ESQUERDA PARA A DIREITA */
{
dist_o_b=0;
/* Calcula a distancia ao veiculo anterior na outra faixa*/
while ((pEstradaDir[(L+i-dist_o_b) % L] == -1) && (dist_o_b<L))
dist_o_b++;
dist_o = 1;
/* Calcula a distancia ao veiculo seguinte na outra faixa*/
while ((pEstradaDir[(i+dist_o) % L] == -1) && (dist_o<L))
dist_o++;
dist = 1;
/* Calcula a distancia ao veiculo seguinte na mesma faixa*/
while (pEstradaEsq[(i+dist) % L] == -1)
dist++;
/* Verifica as condicoes de mudanca da faixa DA ESQUERDA PARA A DIREITA */
bool = (((double)rand()/(RAND_MAX+1)) < probR);
if (
(
Vmax < dist_o_b // manter dist. de segur. ao anterior na outra faixa
)
&&
(
(
(!bool)
&&
(
(
(Vmax + 1 + Voff) < dist_o // existe espaço suficiente na outra faixa
)
&&
(
```

```
(Vmax +1 + Voff) < dist // existe espaço suficiente à frente
)
)
)
||
(
bool
&&
(pEstradaEsq[i] < dist_o) // existe espaço suficiente na outra faixa
)
)
)
{
pEstradaDirAux[i] = pEstradaEsq[i];
NDirAux++;
pEstradaEsqAux[i]=-1;
NEsqAux-;
}
i += dist; /* Avança para o próximo veiculo */
}
}
for (i=0; i<L; i++)
{
pEstradaEsq[i]=pEstradaEsqAux[i];
pEstradaDir[i]=pEstradaDirAux[i];
}
NDir=NDirAux;
NEsq=NEsqAux;
}
```

Anexo D: Código em C para Modelo de 2 Faixas com 2 Tipos de Veículos

```
**************************************************************************

File CALCULAFLUXO.c

#include "calculafluxo.h"
void CalculaFluxo(double densidade, double *FDir, double *FEsq)
{
double
fluxoDir=0., fluxoEsq=0.,
//fluxoTotal=0.,
b=1., i=0.,
fluxoTotalDir=0., fluxoTotalEsq=0.,
varianciaDir=0., varianciaEsq=0.;
/* Chamada das funções que vão efectuar a inicialização das variáveis */
inicializa(densidade);
/* Se o numero de veiculos é zero ou os sitios estão todos ocupados, em
ambas as faixas, o fluxo é zero, caso contrário vai calcular o fluxo */
if ((N !=0) && (N != 2*L))
{
/* Faz um "aquecimento inicial" com L iterações */
for( i=0. ; i<L ; i++ )
{
muda();
evolui(pEstradaEsq,pEstradaEsqAux, NEsq);
evolui(pEstradaDir,pEstradaDirAux,NDir);
```

```
}
/* Faz uma primeira amostragem */
for( i=1. ; i<(L+1) ; i++ )
{
muda();
fluxoTotalEsq = fluxoTotalEsq*(1.-1./i)+evolui(pEstradaEsq,pEstradaEsqAux,NEsq)/i;
fluxoTotalDir = fluxoTotalDir*(1.-1./i)+evolui(pEstradaDir,pEstradaDirAux,NDir)/i;
}
/* gera valores para o fluxo até obter uma boa estatistica, efectuando no minimo
30 iterações para garantir o teorema do limite central */
while (
(b < 30.)
||
(varianciaDir > (b*sqr(ERRO*fluxoTotalDir)))
||
(varianciaEsq > (b*sqr(ERRO*fluxoTotalEsq)))
)
{
b++;
fluxoDir = 0.;
fluxoEsq = 0.;
/* Faz L iterações para o cálculo do fluxo */
for( i=1. ; i<(L+1) ; i++ )
{
muda();
fluxoEsq = fluxoEsq*(1.-1./i)+evolui(pEstradaEsq,pEstradaEsqAux,NEsq)/i;
fluxoDir = fluxoDir*(1.-1./i)+evolui(pEstradaDir,pEstradaDirAux,NDir)/i;
}
fluxoTotalDir = fluxoTotalDir*(1.-1./b)+fluxoDir/b;
varianciaDir = varianciaDir*(b-2.)/(b-1.)+b*(sqr((fluxoDir-fluxoTotalDir)/(b-1.)));
fluxoTotalEsq = fluxoTotalEsq*(1.-1./b)+fluxoEsq/b;
varianciaEsq = varianciaEsq*(b-2.)/(b-1.)+b*(sqr((fluxoEsq-fluxoTotalEsq)/(b-1.)));
}
```

```
}
//fluxoTotal = fluxoTotalDir + fluxoTotalEsq;
*FEsq = fluxoTotalEsq;
*FDir = fluxoTotalDir;
//printf("%s%f\n","fluxo total=", fluxoTotal);
//fprintf(pFileOut, "%s\t%f\n","fluxo total=",fluxoTotal);
printf("%s%f\t%s%f\n","fluxo dir = ", fluxoTotalDir,"fluxo esq = ",fluxoTotalEsq);
fprintf(pFileOut,"%s%f\t%s%f\n","fluxo dir = ", fluxoTotalDir,"fluxo esq = ",fluxoTotalEsq);
//return fluxoTotal;
}
```

****

File calculafluxo.h

```
#include <stdlib.h>
#include <stdio.h>
typedef struct strVeiculo {
char vel;
char tipo;
} VEICULO;
double evolui(VEICULO*, VEICULO*, unsigned int);
void inicializa(double);
void muda(void);
extern unsigned int L, N, NDir, NEsq;
extern VEICULO *pEstradaDir, *pEstradaEsq, *pEstradaDirAux, *pEstradaEsqAux;
extern FILE *pFileOut;
extern double ERRO;
#define sqr(x) ((x)*(x)) /* Função que calcula o quadrado de um número real */
```

****

File Corrente.c

```
/*-----------------------------------------
Module: C:\ARQUIVO\MESTRADO_MA\TRAFEGO\PROF4\CORRENTE.C
```

```
Author: Joaquim
Project: auto-estrada
State: em evolução
Creation Date: 29.12.2000
Description: Tráfego em auto-estrada com duas faixas;
Regras de Mudança de Wagner;
15% pesados e 85% ligeiros;
_____*/
#include "corrente.h"
int main(void)
{
double densidade=0., FDir, FEsq;
float xray[51], yDray[51], yEray[51];
janela(); // gera janela para escolha das características do sistema
alocar(); //reserva memória para os arrays compridos para não ter que repetir
printf("Tamanho= %d\tProb.= %.1f\tVmaxA= %d\t\tErro= %.3f\n\n",L,prob,VmaxA,ERRO);
// escreve cabeçalhos
fprintf(pFileOut,"Tamanho= %d\tProb.= %.1f\tVmaxA= %d\n\n",L,prob,VmaxA);
// Para vários valores de densidade vai determinar o correspondente fluxo
for (unsigned int i = 0; i <51 ; i++)
{
densidade= i/50.;
printf ("%s%.2f\t","densidade = ",densidade);
fprintf(pFileOut, "%s%.2f\t","densidade = ",densidade);
xray[i] = densidade ;
CalculaFluxo(densidade, &FDir, &FEsq);
yDray[i] = FDir;
yEray[i] = FEsq;
}
grafico(xray,yDray,yEray);
return (0);
}
```

****

File Corrente.h

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
/* Variaveis Globais*/
unsigned char VmaxA=4,VmaxB=6,LA=3,LB=1;
typedef struct strVeiculo {
char vel;
char tipo;
}
VEICULO;
VEICULO
*pEstradaDir=NULL,
*pEstradaEsq=NULL,
*pEstradaDirAux=NULL,
*pEstradaEsqAux=NULL;
unsigned int
NDir, NEsq, /* numero de veiculos na faixa da direita e na da esquerda, respectivamente
*/
L, /* numero de sitios na estrada */
N; /* numero de veiculos */
unsigned char VmaxA,VmaxB; /* velocidade maxima */
double prob, /* probabilidade de reduzir a velocidade */
ERRO; // erro relativo
FILE *pFileOut=NULL; // ponteiro para ficheiro de saida de dados
// protótipos:
void alocar(void);
void CalculaFluxo(double,double*,double*);
void grafico(float*,float*,float*);
void janela(void);

*****************************************************************************
```

File CORRENTE_ALOCAR.c

```
#include <stdlib.h>
#include <stdio.h>
typedef struct strVeiculo {
char vel;
char tipo;
} VEICULO;
extern FILE *pFileOut;
extern VEICULO
*pEstradaDir,
*pEstradaEsq,
*pEstradaDirAux,
*pEstradaEsqAux;
extern unsigned int L;
void alocar(void)
{
//.....abre ficheiro de saida de dados.....
if((pFileOut = fopen("corrente.dat", "w")) == NULL)
{
printf("can't open file\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaDir.....
pEstradaDir = malloc(L*sizeof(VEICULO));
if ( pEstradaDir==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaEsq.....
pEstradaEsq = malloc(L*sizeof(VEICULO));
if ( pEstradaEsq==NULL )
{
```

```
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaDirAux.....
pEstradaDirAux = malloc(L*sizeof(VEICULO));
if ( pEstradaDir==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
//.....alocação de pEstradaEsqAux.....
pEstradaEsqAux = malloc(L*sizeof(VEICULO));
if ( pEstradaEsq==NULL )
{
printf("\n erro na atribuicao de memoria\n");
exit(EXIT_FAILURE);
}
}
```

****

File CORRENTE_DISLIN.c

```
#include <stdio.h>
#include <stdlib.h>
#include <dislin.h>
void janela(void);
void FunL(int);
void FunProb(int);
void FunErro(int);
//void FunVmax(int);
static int
idMainLLista,
//idMainVmaxLista,
idMainProbLista,
```

```
idMainErroLista;
static char
listaL[]="10|100|500|1000|1500",
//listaVmax[]="1|2|5|6|10",
listaProb[]=".1|.2|.5",
listaErro[]=".1|.01|.001";
extern unsigned int L;
//extern unsigned char Vmax;
extern double prob, ERRO;
void grafico(float* x,float* y1, float* y2)
{
metafl("cons"); /*define o ficheiro metafile */
page(2970,2100);
disini();
pagfll(-1);
pagera();
complx();
color ("black");
//axspos(450,1800);
//axslen(2100,1200);
name("Densidade","x");
name("Fluxo","y");
ticks(5,"xy");
titlin("Evolucao do Fluxo",1);
titlin("Densidade/Fluxo",3);
graf(0.0,1.0,0.0,0.1,0.0,1.1,0.0,0.1);
title();
color("red");
//incmrk(3);
curve(x,y1,51);
color("green");
//incmrk(4);
curve(x,y2,51);
```

```
//dash();
xaxgit();
/* Guarda o gráfico num ficheiro */
/* rtiff ("exemplo.tif");*/
color("fore");
disfin();
}
////////////////////////////////////////////////////////////////////////////////
void janela(void)
{
int idMain,
idMainL,
idMainProb,
idMainErro
// idMainVmax,
// idMainLTxt,
// idMainVmaxTxt,
// idMainProbTxt,
// idMainErroTxt
;
swgtit("Escolha do sistema");
swgpop("NOOK");
swgpop("nohelp");
swgpop("noquit");
idMain=wgini("VERT"); // cria a janela principal
idMainL=wgbas(idMain,"HORI");
wglab(idMainL,"Tamanho:");
idMainLLista=wgdlis(idMainL, listaL, 2);
FunL(idMainLLista); // para o valor por defeito
swgcbk(idMainLLista,FunL);
// idMainVmax=wgbas(idMain,"HORI");
// wglab(idMainVmax,"Vmax:");
// idMainVmaxLista=wgdlis(idMainVmax, listaVmax, 2);
```

```
// FunVmax(idMainVmaxLista); // para o valor por defeito
// swgcbk(idMainVmaxLista,FunVmax);
idMainProb=wgbas(idMain,"HORI");
wglab(idMainProb,"Probabilidade:");
idMainProbLista=wgdlis(idMainProb, listaProb,1);
FunProb(idMainProbLista); // para o valor por defeito
swgcbk(idMainProbLista,FunProb);
idMainErro=wgbas(idMain,"HORI");
wglab(idMainErro,"Erro relativo:");
idMainErroLista=wgdlis(idMainErro, listaErro,2);
FunErro(idMainErroLista); // para o valor por defeito
swgcbk(idMainErroLista,FunErro);
wgok(idMain);
wgfin();
}
////////////////////////////////////////////////////////////////////////////
void FunL(int id)
{
L=atoi(itmstr(listaL,gwglis(idMainLLista)));
}
void FunProb(int id)
{
prob=atof(itmstr(listaProb,gwglis(idMainProbLista)));
}
//void FunVmax(int id)
//{
// Vmax=atoi(itmstr(listaVmax,gwglis(idMainVmaxLista)));
//}
void FunErro(int id)
{
ERRO=atof(itmstr(listaErro,gwglis(idMainErroLista)));
}

**************************************************************************
```

File Evolui.c

```
/*-------------------------------------------------
Module: C:\ARQUIVO\MESTRADO_MA\TRAFEGO\PROF4\EVOLUI.C
Author: Joaquim
Project: prof4
State: em evolução
Creation Date: 30.12.2000
Description: Rotina responsável pela actualizacao da
posicao dos veiculos numa faixa em cada
iteracao
-------------------------------------------------*/
#include "evolui.h"
/*-------------------------------------------------
Procedure: evolui ID:1
Purpose: faz avançar cada veículo numa dada faixa em
paralelo segundo as regras mais simples de
NaSch
Input: endereço da faixa (pEstrada)
endereço da faixa actualizada
(pEstradaAuxiliar)
número de veículos na faixa (N)
Output: "passam" =
1 -> se passa veículo pelo último sítio
0 -> se NÃO passa veículo pelo último sítio
Errors: nenhum
-------------------------------------------------*/
double evolui(VEICULO *pEstrada,VEICULO *pEstradaAux, unsigned int N)
{
int i, gap, passam=FALSE;
char estrada1,tipo1,Vmax,Comp;
if((N<1) || ((N+1)>L))
return passam;
// Determina o 1º veiculo
```

```
i=0;
while((i<L) && (pEstrada[i].vel == -1))
i++;
while (i < L) /* Procura os sitio da estrada com veiculo e aplica as regras de
actualização da velocidade associada a cada veiculo */
{
estrada1 = pEstrada[i].vel; // velocidade do veículo actual
tipo1 = pEstrada[i].tipo; // tipo do veículo actual
if (tipo1=='A') // determina características do veículo actual
{
Vmax=VmaxA;
Comp=LA-1;
}
else
{
Vmax=VmaxB;
Comp=LB-1;
}
gap = 1;
/* Calcula a distancia ao veiculo seguinte */
while(pEstrada[(i+gap) % L].vel == -1)
gap++;
gap=gap-Comp; // desconta o comprimento da frente do veículo
/* Se a velocidade do veiculo é inferior à velocidade máxima, então a velocidade
que lhe está associada pode ser aumentada de uma unidade */
estrada1 = min(estrada1+1,Vmax);
/* Se a distancia ao veiculo seguinte é inferior ou igual à velocidade actual do veiculo,
então a sua velocidade tem que ser reduzida e passa a ter o valor da distancia ao
veiculo seguinte menos uma unidade, para que dessa forma não haja colisão */
estrada1 = min(gap-1,estrada1);
/* Com uma probabilidade definida "prob" e se a velocidade for superior a zero,
a velocidade do veiculo é diminuida de uma unidade */
if (((double)rand()/(RAND_MAX+1)) < prob)
```

```
estrada1 = max(estrada1-1,0);
pEstradaAux[i].vel=-1; // retira veículo
pEstradaAux[i].tipo=0;
pEstradaAux[(i+estrada1) % L].vel=estrada1; // coloca veículo
pEstradaAux[(i+estrada1) % L].tipo=tipo1;
/* Se há a passagem de um veiculo no local considerado o inicio da estrada,
a variável passam toma o valor 1, para o cálculo do fluxo */
if (i+estrada1 > L-1)
passam = TRUE;
i = i+gap+Comp; // posição do vaículo da frente
}
for (i=0; i<L; i++) // actualiza as posições na faixa
pEstrada[i]=pEstradaAux[i];
return(passam);
}
```

****

File evolui.h

```
#include <stdlib.h>
typedef struct strVeiculo {
char vel;
char tipo;
} VEICULO;
extern unsigned int L;
extern unsigned char VmaxA,VmaxB,LA,LB;
extern double prob;
#define min(x,y) (((x)<(y))?(x):(y)) // Função que calcula o minimo de dois números
inteiros
#define max(x,y) (((x)>(y))?(x):(y)) // Função que calcula o máximo
static double TRUE=1.0, FALSE=0.0;
```

****

File INICIALIZA.c

```
/*-----------------------------------------------------
Module: C:\ARQUIVO\MESTRADO_MA\TRAFEGO\PROF4\INICIALIZA.C
Author: Joaquim
Project: prof4
State: em desenvolvimento
Creation Date: 20.12.2000
Description: Inicializa a "estrada" e dá-lhe uma
configuração inicial alinhada dos
veiculos , com velocidade nula
-----------------------------------------------------*/
#include "inicializa.h"
void inicializa(double densidade)
{
unsigned int i,
nPDir, // # de pesados na faixa direira
nPEsq;
IniciaParam(densidade); /* no contexto de veículos de comprimentos não unitários N é
o # de sítios ocupados e não o # de veículos*/
//.....
/* Determina quantos sitios, em cada faixa, vão estar ocupados */
NDirAux=NDir = N/2;
NEsqAux=NEsq = N - NDir;
nPDir = NDir*.15/(.15*3+.85); // # de pesados na direita
nPEsq = NEsq*.15/(.15*3+.85);
//.....
/* Distribuição dos veiculos pela faixa da direita */
/* Atribui a todos os sitios o valor -1, ou seja, sem veiculos */
for( i=0 ; i<L ; i++ )
{
pEstradaDir[i].vel = -1;
pEstradaDir[i].tipo = 0;
}
/* Aos NDir sitios calculados, atribui-lhe um veiculo, com velocidade nula */
```

```
for( i=0 ; i<nPDir*3 ; i+=3 ) // a morada de um pesado é o sítio traseiro!
{
pEstradaDir[i].vel = 0;
pEstradaDir[i].tipo = 'A';
}
for( i=nPDir*3 ; i<NDir ; i++ )
{
pEstradaDir[i].vel = 0;
pEstradaDir[i].tipo = 'B';
}
/* Dos L sitios da faixa da direita, vai extrair NDir, que vao ser os que vão conter veiculos
*/
// permuta(pEstradaDir,NDir); // pois é: agora já não se pode distribuir os carros!!!
//.....
/* Distribuição dos veiculos pela faixa da esquerda */
/* Atribui a todos os sitios o valor -1, ou seja, sem veiculos */
for( i=0 ; i<L ; i++ )
{
pEstradaEsq[i].vel = -1;
pEstradaEsq[i].tipo = 0;
}
/* Aos NEsq sitios calculados, atribui-lhe um veiculo, com velocidade nula */
for( i=0 ; i<nPEsq*3 ; i+=3 )
{
pEstradaEsq[i].vel = 0;
pEstradaEsq[i].tipo = 'A';
}
for( i=nPEsq*3 ; i<NEsq ; i++ )
{
pEstradaEsq[i].vel = 0;
pEstradaEsq[i].tipo = 'B';
}
```

```
/* Dos L sitios da faixa da esquerda, vai extrair NEsq, que vao ser os que vão conter veiculos
*/
// permuta(pEstradaEsq,NEsq);
for (i=0; i<L; i++)
{
pEstradaEsqAux[i]=pEstradaEsq[i];
pEstradaDirAux[i]=pEstradaDir[i];
}
}
```

****

File inicializa.h

```
/*-----------------------------------------------
Module: C:\ARQUIVO\MESTRADO_MA\TRAFEGO\PROF4\inicializa.h
Author: Joaquim
Project: prof4
State: em desenvolvimento
Creation Date: 20.12.2000
Description: cabeçalhos para "inicializa.c"
-----------------------------------------------*/
#include <stdlib.h>
typedef struct strVeiculo {
char vel;
char tipo;
} VEICULO;
extern VEICULO *pEstradaDir, *pEstradaEsq, *pEstradaEsqAux, *pEstradaDirAux;
extern unsigned int NDir, NEsq, L, N;
extern unsigned char VmaxA, VmaxB;
unsigned int NDirAux, NEsqAux;
#define min(x,y) (((x)<(y))?(x):(y)) // Função que calcula o minimo de
void permuta(VEICULO*,unsigned int);
void IniciaParam(double);
```

****

File muda.c

```
/*-----------------------------------------------------------
Module: C:\ARQUIVO\MESTRADO_MA\TRAFEGO\PROF4\MUDA.C
Author: Joaquim
Project: prof4
State: em desenvolvimento
Creation Date: 20.12.200
Description: Rotina responsável pela mudança de faixa
-----------------------------------------------------------*/
#include "muda.h"
char Voff=8;
double probR=.05;
void muda(void)
{
int i,
dist_o_b=0, //distancia ao veiculo anterior na outra faixa
dist, //distancia ao veiculo seguinte na mesma faixa
dist_o; //distancia ao veiculo seguinte na outra faixa
char bool, tipo1, Vmax, Comp, Vmax_o_b;
/* Se o número de veiculos na faixa da direita é diferente de zero, aplica as
regras de mudanca de faixa DA DIREITA PARA A ESQUERDA */
if (NDir > 0)
{
i = 0; /* Determina o 1º veiculo na faixa da direita*/
while(pEstradaDir[i].vel == -1)
i++;
while (i < L) /* Procura os sitios da estrada com veiculo e aplica as regras de
mudança de faixa DA DIREITA PARA A ESQUERDA */
{
tipo1 = pEstradaDir[i].tipo;
if (tipo1=='A')
{
Vmax=VmaxA;
```

```
Comp=LA-1;
}
else
{
Vmax=VmaxB;
Comp=LB-1;
}
dist_o_b=0; /* Calcula a distancia ao veiculo anterior na outra faixa */
while ((pEstradaEsq[(L+i-dist_o_b) % L].vel == -1) && (dist_o_b<L))
dist_o_b++;
Vmax_o_b=VmaxB;
if (pEstradaEsq[(L+i-dist_o_b) % L].tipo == 'A')
{
dist_o_b -= (LA - 1);
Vmax_o_b = VmaxA;
}
dist = 1; /* Calcula a distancia ao veiculo seguinte na mesma faixa */
while (pEstradaDir[(i+dist) % L].vel == -1)
dist++;
dist -= Comp;
dist_o = 1; /* Calcula a distancia ao veiculo seguinte na outra faixa */
while ((pEstradaEsq[(i+dist_o) % L].vel == -1) && (dist_o<L))
dist_o++;
dist_o -= Comp;
/* Verifica as condicoes de mudança da faixa DA DIREITA PARA A ESQUERDA */
if (
(Vmax_o_b < dist_o_b) // manter dist. de segur. ao anterior na outra faixa
&&
((Vmax +1)> dist) // nesta faixa não posso ir tão rápido como desejo
&&
((dist_o +1)> dist) // a outra faixa não é pior
)
{
```

```
pEstradaEsqAux[i] = pEstradaDir[i];
NEsqAux +=(Comp+1);
pEstradaDirAux[i].vel=-1;
pEstradaDirAux[i].tipo=0;
NDirAux -=(Comp+1);
}
i += (dist+Comp); /* Avança para o próximo veiculo */
}
}
/* Se o numero de veiculos na faixa da esquerda é diferente de zero, aplica as
regras de mudanca de faixa DA ESQUERDA PARA A DIREITA */
if (NEsq > 0 )
{
i = 0; /* Determina o 1º veiculo na faixa da esquerda*/
while(pEstradaEsq[i].vel == -1)
i++;
while (i < L) /* Procura os sitios da estrada com veiculo e aplica as regras de
mudança de faixa DA ESQUERDA PARA A DIREITA */
{
tipo1 = pEstradaEsq[i].tipo;
if (tipo1=='A')
{
Vmax=VmaxA;
Comp=LA-1;
}
else
{
Vmax=VmaxB;
Comp=LB-1;
}
dist_o_b=0; /* Calcula a distancia ao veiculo anterior na outra faixa*/
while ((pEstradaDir[(L+i-dist_o_b) % L].vel == -1) && (dist_o_b<L))
dist_o_b++;
```

```
Vmax_o_b=VmaxB;
if (pEstradaDir[(L+i-dist_o_b) % L].tipo == 'A')
{
dist_o_b -= (LA - 1);
Vmax_o_b = VmaxA;
}
dist_o = 1; /* Calcula a distancia ao veiculo seguinte na outra faixa*/
while ((pEstradaDir[(i+dist_o) % L].vel == -1) && (dist_o<L))
dist_o++;
dist_o -= Comp;
dist = 1; /* Calcula a distancia ao veiculo seguinte na mesma faixa*/
while (pEstradaEsq[(i+dist) % L].vel == -1)
dist++;
dist -= Comp;
/* Verifica as condicoes de mudanca da faixa DA ESQUERDA PARA A DIREITA */
bool = (((double)rand()/(RAND_MAX+1)) < probR);
if (
(Vmax_o_b < dist_o_b) // manter dist. de segur. ao anterior na outra faixa
&&
(
(
(!bool)
&&
(
((Vmax + 1 + Voff) < dist_o) // existe espaço suficiente na outra faixa
&&
((Vmax +1 + Voff) < dist) // existe espaço suficiente à frente
)
)
||
(
bool
&&
```

```
(pEstradaEsq[i].vel < dist_o) // existe espaço suficiente na outra faixa
)
)
)
{
pEstradaDirAux[i] = pEstradaEsq[i];
NDirAux +=(Comp+1);
pEstradaEsqAux[i].vel=-1;
pEstradaEsqAux[i].tipo=0;
NEsqAux -=(Comp+1);
}
i += (dist+Comp); /* Avança para o próximo veiculo */
}
}
for (i=0; i<L; i++)
{
pEstradaEsq[i]=pEstradaEsqAux[i];
pEstradaDir[i]=pEstradaDirAux[i];
}
NDir=NDirAux;
NEsq=NEsqAux;
}
```

****

File muda.h

```
/*_____________________________________________
Module: C:\ARQUIVO\MESTRADO_MA\TRAFEGO\PROF4\muda.h
Author: Joaquim
Project: prof4
State: em desenvolvimento
Creation Date: 20.12.20
Description: cabeçalho para "muda.c"
_____________________________________________*/
```

```
#include <stdlib.h>
typedef struct strVeiculo {
char vel;
char tipo;
} VEICULO;
extern unsigned int NDir, NEsq, L, NDirAux, NEsqAux;
extern VEICULO *pEstradaDir, *pEstradaEsq,
*pEstradaDirAux, *pEstradaEsqAux;
extern char VmaxA,VmaxB,LA,LB;

*****************************************************************************

File ROTINASPEQUENAS.c

/*-----------------------------------------------
Module: C:\ARQUIVO\MESTRADO_MA\TRAFEGO\PROF4\ROTINASPEQUENAS.C
Author: Joaquim
Project: prof4
State:
Creation Date:
Description: algumas rotinas auxiliares
-----------------------------------------------*/
#include <stdlib.h>
typedef struct strVeiculo {
char vel;
char tipo;
} VEICULO;
extern unsigned int L, N;
void permuta(VEICULO *V, unsigned int r) // de L elementos iniciais vai extrair r, aleatoriamente
{
int k, u;
VEICULO troca;
for( k=L-1 ; k>L-r-1 ; k–)
{
```

```
u = k*(double) rand()/(RAND_MAX+1);
troca = V[u];
V[u] = V[k];
V[k] = troca;
}
}
////////////////////////////////////////////////////////////////////////////////
void IniciaParam(double densidade) // Inicializa os parametros (Variáveis)
{
N = densidade*2*L; /* no contexto de veículos de comprimentos não unitários N é
o # de sítios ocupados e não o # de veículos*/
}
```