Notes for simulation of traffic flow on an arbitrary network of one-way single-lane roads with traffic lights at intersections.

Charles S. Peskin
April 17, 2017
C = car index, nc = # of cars

i = intersection index, ni = # of intersections

b = block index, nb = # of blocks

i1(b), i2(b) = indices of intersections connected by block b, ordered by the direction of traffic flow. (All blocks are one-way.)

nbini(i) = # of blocks entering intersection i

bin(i,j) = index of jth block entering intersection i  
\[ j = 1 \ldots nbini(i) \]

nbout(i) = # of blocks leaving intersection i

bout(i,j) = index of jth block leaving intersection i  
\[ j = 1 \ldots nbout(i) \]
Note that nbin, bin can be derived from i2, and that nout, bout can be derived from i1, as follows:

```
for i = 1:ni
    nbin(i) = sum (i2 == i)
    nbout(i) = sum (i1 == i)
end

nbinmax = max (nbin)
nboutmax = max (nbout)
bin = zeros (ni, nbinmax)
bout = zeros (ni, nboutmax)
for i = 1:ni
    bm(i, 1:nbin(i)) = find (i2 == i)
    bout(i, 1:nbout(i)) = find (i1 == i)
end
```

As a check, it should be the case that

\[ \text{sum(nbin)} = \text{sum(nbout)} = nb \]
Traffic lights

At any given time, the traffic light at intersection i is green for exactly one of the blocks that enter that intersection and red for all of the others entering that intersection.

Let \( j_{\text{green}}(i) \) be an integer designating which block has the green light, where

\[
1 \leq j_{\text{green}}(i) \leq n_{\text{bin}}(i)
\]

Let \( S(b) \) be the state of the light at the end of block \( b \), where \( S=0 \) denotes red and \( S=1 \) denotes green.

Given the array \( j_{\text{green}} \), \( S \) can be set as follows:

\[
S = \text{zeros}(1, n_{\text{b}})
\]

\[
\text{for } i = 1 : n_i
\]

\[
b = \text{bin}(i, j_{\text{green}}(i))
\]

\[
S(b) = 1
\]

end
Geometric Information about the network of roads

\( (x_i(i), y_i(i)) = \) coordinates of intersection \( i \)

\( L(b) = \) length of block \( b \)

\( (u_x(b), u_y(b)) = \) unit vector along block \( b \) in direction of traffic flow

Given \( x_i, y_i \), we can find \( L, u_x, u_y \) as follows

\[
\begin{align*}
u_x &= x_i(i_2) - x_i(i_1) \\
u_y &= y_i(i_2) - y_i(i_1)
\end{align*}
\]

\[
L = \sqrt{u_x \cdot A_2 + u_y \cdot A_2}
\]

\[
\begin{align*}
u_x &= u_x / L \\
u_y &= u_y / L
\end{align*}
\]
Cars on blocks

Let \( p(c) \) be the position of car \( c \) on whatever block it happens to be on, measured as distance from the start of the block. If car \( c \) is on block \( b \), then

\[
0 \leq p(c) < L(b)
\]

and the coordinates of car \( c \) are given by

\[
\begin{align*}
x(c) &= x_i(iL(b)) + p(c) \times u_x(b) \\
y(c) &= y_i(iL(b)) + p(c) \times u_y(b)
\end{align*}
\]

To access all of the cars on a block in order of decreasing \( p \), we use the following linked-list data structure:

\[
\begin{align*}
\text{firstcan}(b) &= \text{index of first car on block } b \\
\text{nextcan}(c) &= \text{index of car immediately behind } c \text{ on the same block} \\
\text{lastcan}(b) &= \text{index of last car on block } b
\end{align*}
\]

In all cases, an entry of 0 means that there is no such car. Thus \( \text{nextcan}(\text{lastcan}(b)) = 0 \), and if block \( b \) is empty then \( \text{firstcan}(b) = \text{lastcan}(b) = 0 \).
Entry of cars and choice of their destinations

Cars enter the roadway (from parking garages or parking spaces) at random times and locations. Let \( R \) be the rate at which this occurs. Then \( R \) has units of \( 1/(\text{time} \times \text{length}) \). Choose the time step \( dt \) small enough that \( R \times L_{\text{max}} \times dt << 1 \), where \( L_{\text{max}} \) is the largest length of any block. Then we can make the approximation that at most one car enters the roadway per block per time step. To decide whether this happens and to choose the location \( p \) on the block if it does, we can do the following for each block \( b \):

\[
\text{if } (\text{rand} < dt \times R \times L(b))
\]
\[
nc = nc + 1
\]
\[
p(nc) = \text{rand} \times L(b)
\]

End

When a car enters the roadway, it is assigned a destination. This can also be done randomly. Let \( bd(c) \) be the block on which the destination lies and let \( pd(c) \) be the position on that block, expressed as distance from the
start of the block. A simple way to make this choice is

\[ bd(c) = 1 + \text{floor}(\text{rand} \times nb) \]
\[ pd(c) = \text{rand} \times L(bd(c)) \]

but note that this choice gives equal weight to any block regardless of its length. To make the probability of choosing a block be proportional to its length, we can use the method of rejection:

\[ bd(c) = 1 + \text{floor}(\text{rand} \times nb) \]
\[ pd(c) = \text{rand} \times L_{\text{max}} \]
\[ \text{while } (pd(c) \geq L(bd(c))) \]
\[ \quad \begin{align*}
\quad bd(c) &= 1 + \text{floor}(\text{rand} \times nb) \\
\quad pd(c) &= \text{rand} \times L_{\text{max}}
\end{align*} \]
\[ \text{end} \]

In this version we keep trying until we find a position that fits on the block, and this makes the block that is ultimately chosen more likely to be a longer one. In fact, the probability of choosing a block is exactly proportional to its length, and \( pd \) is uniformly distributed over that length.
Steering a car to its destination (despite one-way streets!)

For this we need the Cartesian coordinates of the destination, which are given by

\[
\begin{align*}
xd(c) &= xi(i1(bd(c))) + pd(c) \times UX(bd(c)) \\
yd(c) &= yi(i1(bd(c))) + pd(c) \times UY(bd(c))
\end{align*}
\]

When a car comes to an intersection, it can choose to enter any of the blocks leaving that intersection. The natural choice is the one that most nearly points towards the destination. To determine this, evaluate the vector from the intersection to the destination, and then the dot product of that vector with all of the unit vectors of the blocks leaving the intersection. The block that should be chosen is the one that maximizes this dot product (in the algebraic sense, i.e., choose the most positive or least negative result, not the one largest in magnitude).
According to the above prescription, if car $c$ is at intersection $i$, it should choose the next block $b$ to enter in the following way:

\[ \begin{align*}
    xdvec &= xd(c) - xi(i) \\
    ydvec &= yd(c) - yi(i)
\end{align*} \]

\[ dp = u_x(b_{out}(i, 1:n_{b_{out}(i)}) \times \text{xdvec} + u_y(b_{out}(i, 1:n_{b_{out}(i)}) \times \text{ydvec} \]

\[ [dp_{max}, jb] = \max(dp) \]

\[ b = b_{out}(i, jb) \]

In the above use of max, there are two outputs. The second one, $jb$, is the index of the element of $dp$ that has the maximum value.

The above steering algorithm works well for reasonable road networks, including cases in which it is necessary to go around the block to reach the destination because of one-way streets, but it is not guaranteed to work. For some roadway layouts and some destinations, a car can get trapped and go through a cycle of
blocks repeatedly by following the above algorithm without ever reaching its destination. One way to avoid this is for the can to remember the intersections it has been to and the choices it has made there, and never make the same choice twice at any given intersection. Another way that is easier to program is for the can to decide randomly at each intersection whether to follow the above algorithm or to choose a random block. This can be programmed as follows:

```plaintext
if (rand < prchoice)
    jb = 1 + floor (rand * n bout (i))
    b = bout (i, jb)
else
    choose b by the method of maximizing the dot product as described above

Here prchoice is the probability that a random choice will be made.
```
% Main program: traffic.m

initialize
for clock = 1: clockmax
    t = clock * dt
    setlights
    createcars
    movecars
    plotcars
end

% setlights.m
if t > tlc
    for i = 1: ni
        jgreen(i) = jgreen(i) + 1
        if jgreen(i) > nbins(i)
            jgreen(i) = 1
        end
    end
end
tlc = tlc + tlesstep
end
s = zeros(1, nb)
for i = 1: ni
    b = bin(i, jgreen(i))
    s(b) = 1
end
% initialization

jgreen = ones(1, ni)
tlastep = % time interval between light changes
tlc = tlastep

% createcars.m
for b = 1: nb
    if (rand < dt*R*L(b))
        nc = nc + 1
        p(nc) = rand*L(b)
        x(nc) = xi(i1(b)) + p(nc)*ux(b)
        y(nc) = yi(i1(b)) + p(nc)*uy(b)
        onroad(nc) = 1
        insert new car
        choose destination
        nextb(nc) = b
        tenter(nc) = t
        benter(nc) = b
        penter(nc) = p(nc)
    end
end
% insertnewcan.m
C = firstcan(b)
if (c == 0 || p(nc) > p(c))
    nextcan(nc) = c
    firstcan(b) = nc
    if (c == 0)
        lastcan(b) = nc
    end
else if p(nc) <= p(lastcan(b))
    nextcan(lastcan(b)) = nc
    lastcan(b) = nc
else
    CA = C
    C = nextcan(c)
    while (p(nc) <= p(c))
        CA = C
        C = nextcan(c)
    end
    nextcan(CA) = nc
    nextcan(nc) = c
end
% choose destination
% use method of rejection to choose a block with probability proportional to
% its length, and with p uniformly distributed in that block.

bd(nc) = 1 + floor(rand * nb)
pd(nc) = rand * Lmax

while (pd(nc) >= L(bd(nc)))
    bd(nc) = 1 + floor(rand * nb)
pd(nc) = rand * Lmax
end

xd(nc) = xi(i1(bd(nc)) + pd(nc)*ux(bd(nc))
yd(nc) = yi(i1(bd(nc)) + pd(nc)*uy(bd(nc))

% Lmax = max(L)
function b = 1 : nb
    c = firstcan(b)
    while (c > 0)
        if (c == firstcan(b))
            if (bd(c) == b) && (pd(c) > p(c))
                d = dmax
            elseif (S(b) == 0)
                d = L(b) - p(c)
            else
                decide next block
                if (lastcan(nextb(c)) > 0)
                    d = (L(b) - p(c)) + p(lastcan(nextb(c)))
                else
                    d = dmax
                end
            end
        else
            d = p(ca) - p(c)
        end
    end
    pz = p(c); nextc = nextcan(c)
    p(c) = p(c) + dt * v(d)
\[
\begin{align*}
\text{if}(b = bd(c)) \land \text{if}(p_z < pd(c)) \land \text{if}(pd(c) \leq p(c)) & \quad \text{remove car} \\
\text{elseif} \ (L(b) \leq p(c)) & \quad p(c) = p(c) - L(b) \\
\text{if}(nextb(c) = bd(c)) \land \text{if}(pd(c) \leq p(c)) & \quad \text{remove car} \\
\text{else} & \quad \text{cart to next block} \\
\text{end} \\
\text{else} & \quad x(c) = x_i(l_1(b)) + p(c) \times ux(b) \\
& \quad y(c) = y_i(l_1(b)) + p(c) \times uy(b) \\
& \quad ca = \_\_ \_ \_ \_ c \\
\text{end} \\
& \quad c = nextc \quad \% \text{ saved value of nextcar}(c) \\
\text{end} \% \text{ while loop over cars on block} \\
\text{end} \% \text{ for loop over blocks}
\end{align*}
\]
% decide next block
if nextb(c) == b
    i = i2(b)
    if rand < p
        jnext = 1 + floor(rand * nbout(i))
        nextb(c) = bout(i, jnext)
    else
        xdvec = xd(c) - xi(i)
        ydvec = yd(c) - yi(i)
        dp = ux(bout(i, 1:nbout(i))) * xdvec + uy(bout(i, 1:nbout(i))) * ydvec
        [dpmax, jnext] = max(dp)
        nextb(c) = bout(i, jnext)
    end
end
% remove car m
onroad(c) = 0; texit(c) = t
if (c == firstcan(b))
    firstcan(b) = nextcan(c)
    if (c == lastcan(b))
        lastcan(b) = 0
end
else
    nextcan(ca) = nextcan(c)
    if (c == lastcan(b))
        lastcan(b) = ca
end
end

% not really needed, but ...
x(c) = xd(c)
y(c) = yd(c)
nextcan(c) = 0
% recall that we previously set nextc = nextcan(c)
\% carto next block. in
\texttt{firstcan(b) = nextcan(c)}
\texttt{if(c == lastcan(b))}
\texttt{lastcan(b) = 0}
\texttt{end}
\texttt{if(lastcan(nextb(c)) == 0)}
\texttt{firstcan(nextb(c)) = c}
\texttt{else}
\texttt{nextcan(lastcan(nextb(c))) = c}
\texttt{end}
\texttt{lastcan(nextb(c)) = c}
\texttt{nextcan(c) = 0}
\% this is why we previously set nextc = nextcan(c)

\( p(c) = p(c) - L(b) \)

\( x(c) = x_i(i \in \text{nextb(c)}) + p(c) \times u_x \text{(nextb(c))} \)
\( y(c) = x_i(i \in \text{nextb(c)}) + p(c) \times u_y \text{(nextb(c))} \)
% plotcars.m

if (nc > 0 && sum(onroad) > 0)
    set(hcars, 'xdata', x(find(onroad)), ...
        'ydata', y(find(onroad)))
end