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.)

$nbin(i)$ = # of blocks entering intersection i

$bin(i,j)$ = index of j^{th} block entering intersection i
$j = 1 \ldots nbin(i)$

$nbout(i)$ = # of blocks leaving intersection i

$bout(i,j)$ = index of j^{th} 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
    bin(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

sum(nbin) = 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 $jgreen(i)$ be an integer designating which block has the green light, where

$$1 \le jgreen(i) \le nbin(i)$$

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 jgreen, s can be set as follows:

```
s = zeros(1, nb)
for i = 1:ni
    b = bin(i, jgreen(i))
    s(b) = 1
end
```

Geometric information about the network of roads

$xi(i), yi(i) =$ coordinates of intersection i

$L(b) =$ length of block b

$(ux(b), uy(b)) =$ unit vector along block b
in direction of traffic flow

Given xi, yi, we can find L, ux, uy
as follows

$ux = xi(i2) - xi(i1)$
$uy = yi(i2) - yi(i1)$

$L = sqrt(ux.^2 + uy.^2)$

$ux = ux./L$
$uy = uy./L$

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

$$x(c) = xi(i1(b)) + p(c) * ux(b)$$
$$y(c) = yi(i1(b)) + p(c) * uy(b)$$

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

firstcar(b) = index of first car on block b
nextcar(c) = index of car immediately behind car c on the same block
lastcar(b) = index of last car on block b

In all cases, an entry of 0 means that there is no such car. Thus nextcar(lastcar(b))=0, and if block b is empty then firstcar(b)= lastcar(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 $1/(\text{time} \cdot \text{length})$. Choose the time step dt small enough that $R * L_{max} * dt << 1$, where L_{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:

```
if (rand < dt*R*L(b))
    nc = nc+1
    p(nc) = rand*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 + floor(rand*nb)
pd(c) = rand*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 + floor(rand*nb)
pd(c) = rand * Lmax
while (pd(c) >= L(bd(c)))
    bd(c) = 1 + floor(rand*nb)
    pd(c) = rand * Lmax
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 be 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

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

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

```
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,jb] = max(dp)

b = bout(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 car 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 car to decide randomly at each intersection whether to follow the above algorithm or to choose a random block. This can be programed as follows:

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

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) > nbin(i)
              jgreen(i) = 1
      end
   end
   tlc = tlc + tlcstep
end
s = zeros(1, nb)
for i = 1:ni
   b = bin(i, jgreen(i))
   s(b) = 1
end
```

```
% initialization for setlights

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


% 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
        insertnewcar
        choosedestination
        nextb(nc) = b
        tenter(nc) = t
        benter(nc) = b
        penter(nc) = p(nc)
   end
end
```

```
% insertnewcar.m
c= firstcar(b)
if (c==0 || p(nc)>p(c))
    nextcar(nc)=c
    firstcar(b)=nc
    if (c==0)
        lastcar(b)=nc
    end
elseif p(nc) <= p(lastcar(b))
    nextcar(lastcar(b)) = nc
    lastcar(b)=nc
    nextcar(nc) = 0
else
    ca=c
    c = nextcar(c)
    while (p(nc) <= p(c))
        ca = c
        c = nextcar(c)
    end
    nextcar(ca)=nc
    nextcar(nc)=c
end
```

```
% choosedestination.m
% use method of rejection to choose a
%  block with probability proportional to
%  its length, and with p uniformly
%  distributed on 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)
```

```
% movecars.m
for b = 1:nb
  c = firstcar(b)
  while (c>0)
    if (c== firstcar(b))
      if (bd(c)==b) && (pd(c) > p(c))
        d = dmax
      elseif (S(b)==0)
        d = L(b)-p(c)
      else
        decidenextblock
        if (lastcar(nextb(c)) > 0
          d= (L(b)-p(c)) + p(lastcar(nextb(c)))
        else
          d = dmax
        end
      end
    else
      d= p(ca) - p(c)
    end
    pz = p(c); nextc = nextcar(c)
    p(c) = p(c) + dt * v(d)
```

```
if(b==bd(c))&&(pz<pd(c))&&(pd(c)<=p(c))
        removecar
elseif(L(b)<=p(c))
        p(c)=p(c)-L(b)
        if(nextb(c)==bd(c))&&(pd(c)<=p(c))
            removecar
        else
            cartonextblock
        end
else
        x(c)=xi(i1(b))+p(c)*ux(b)
        y(c)=yi(i1(b))+p(c)*uy(b)
        ca = ~~[struck]~~ c
end
    c=nextc   % saved value of nextcar(c)
  end % while loop over cars in block
end % for loop over blocks
```

```
% decidenextblock.m
% only do this if decision is not already made
if nextb(c)==b
    i = i2(b)
    if rand < prchoice
        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
```

```
% removecar.m
onroad(c)=0 ; texit(c)=t
if(c== firstcar(b))
    firstcar(b) = nextcar(c)
    if (c==lastcar(b))
        lastcar(b)=0
    end
else
    nextcar(ca)=nextcar(c)
    if(c==lastcar(b))
        lastcar(b) = ca
    end
end

% not really needed, but...
x(c) = xd(c)
y(c) = yd(c)
nextcar(c) =0
% recall that we previously set nextc=nextcar(c)
```

```
% cartonextblock.m
firstcar(b) = nextcar(c)
if (c==lastcar(b))
    lastcar(b)=0
end
if(lastcar(nextb(c))==0)
    firstcar(nextb(c)) = c
else
    nextcar(lastcar(nextb(c)))=c
end
lastcar(nextb(c)) = c
nextcar(c)=0
% this is why we previously set nextc=nextcar(c)
```

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

```
x(c)=xi(i1(nextb(c)))+p(c)*ux(nextb(c))
y(c)=xi(i1(nextb(c)))+p(c)*uy(nextb(c))
```

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