2015 International Conference on Computational Intelligence and Communication Networks

Optimal Route Searching in Networks with Dynamic Weights using Flow Algorithms

Sunit Singh
BITS Pilani Goa Campus
Email: suneet141290@gmail.com

Ram Prasad Joshi
BITS Pilani Goa Campus
Email: rsj@goa.bits-pilani.ac.in

Harsh Kohli
BITS Pilani Goa Campus
Email: harsh14791@gmail.com

***Abstract*—A network with dynamic weights implies a set of vertices interconnected by a set of edges, each of which bears a weight that changes with time. One example of such networks is a traffic network, wherein, the structure of the graph remains constant but the weight on the edges, signifying the amount of traffic (traffic density) changes over time. We have dealt with scenarios where flow algorithm needs to run repeatedly to establish flows in a network with timely changing capacities and we have sought to obtain some form of computational intelligence on that subject. We have aligned our work to the context of traffic networks in order to explore practical inspection of the same. However, this study applies equally for any generic network with continuously changing capacities which requires flow re-setting time and again.**

***Keywords* : Ford Fulkerson Approach, Edmonds-Karp Algorithm, Minimum Cut-Maximum Flow, Traffic Flow**

I. Introduction

For a city map with its roadway network along with its individual edge lengths, it is trivial to find the shortest path from a source point to a destination point. But, given the same network with traffic flow on it, the problem of finding the optimal path opens various avenues of discussion. To name a few - the shortest path in terms of distance, the least time consuming path, the path with least frequent traffic jams. It is noteworthy that the least time consuming path need not always conform to the path with least frequency of traffic jams. Therefore, for a traffic scenario mimicking incremental network behavior, wherein the edge weights signifying the traffic densities at any point in time vary with time, the choice of the most favorable path can stand several interpretations each with their boons and caveats.

We intend to draw a comparison between three approaches that exercise adaptations of the Edmonds-Karp algorithm for the implementation of the Ford Fulkerson method to achieve maximum flow across a network. We then put forth a set of conditions that help in making scenarios that employ repeated implementations of flow algorithms over time, computationally efficient.

With respect to the flow problems paradigm, traffic monitoring approaches discussed here differ in the choice of metric used to grade the source to destination paths in order to pick the subsequent path to induce flow. Since the network's incremental behavior is with respect to time, the flows established to draw inferences from would also need to refurnish with time. This suggests a perpetually repeating implementation of the flow algorithm to model the traffic scenario. Whereupon, our proposed set of conditions, to dictate the re-running of the flow algorithm only when certain criterion is met, comes into operation.

Thus, this study is loosely a conjunction of two segments, the first which attempts to model time-varying traffic situations through adaptations of flow algorithms, and the second which explores improvements for scenarios employing continuous running of algorithms based on Ford Fulkerson method to monitor flow scenarios.

In generic network flow model, the maximum flow between a source and a sink can be determined. The *incremental* nature of the network stems from the timely changing capacities in it, due to which the flow algorithm would need to be triggered each time the set of capacities changed values.

A. Definitions

By *incremental* graphs here, we refer to graphs with dynamic set of weights, that is, the structure of the graph remains as is, but the weighs on the edges change over time. Note that, the word 'increment' has nothing to do with *increase* in the dynamic weights. It is simply to describe graphs whose state changes over time, because its weights re-adjust with time.

Pertaining to the traffic network scenario, we would like to state certain terms that will be used repeatedly through this study. From the point of view of flow algorithms, an augmenting path is a path from source to sink whose edges have capacity to accommodate flow through the path.

In context of our adaptation of the flow algorithm, *grading* an augmenting path will mean prioritizing the path over others while inducing flow in the subsequent step of the algorithm. This prioritizing shall be done with the help of 2 metrics, namely, length of path from source to destination and estimated time of traversal. We would like to make a denotation for that particular augmenting path, which at the current instant is graded best based on any one of the two aforesaid metrics, and offers leverage over other source to destination augmenting paths by calling this the *current-prime* path.

A source to destination path will comprise of several *edge routes*, whereby we mean a path can be disintegrated into edges that could be distinguished from one another by several means. For instance, crossroads, traffic stop lights, road demar-

978-1-5090-0076-0/15 $31.00 © 2015 IEEE
DOI 10.1109/CICN.2015.37

cations etc. Finally, for shorthand, henceforth we will simply regard a source to destination traversal as $s-t$ traversal.

B. Three Regime Model

The Three Regime model proposed by [1] is a macroscopic traffic model that handles traffic parameters like traffic influx and outflow, average vehicle speed, vehicle density(vehicles/mile), inter-vehicle spacing and vehicle flow (vehicles/hour) in 3 regimes, namely; free-flow traffic, congested regime and moving traffic jams. For any general traffic model, these quantities form the basis for the fundamental diagram of traffic flow. The fundamental diagram of traffic flow is a diagram that gives a relation between the traffic flux (vehicles/hour) and the traffic density (vehicles/mile).The regime identification for a particular instance of traffic at some location depends upon the vehicle density at that location. We will take averaged values of vehicle densities over the entire edge while assigning the edges their density values.

Three Regime Models are more intuitive than their single regime counterparts because through piecewise definition of traffic parameters over the entire density range they are able to overcome the problem of defining a single fit which may hold reasonable for a specific set of values of traffic densities and not others.

It is important to note preliminarily, that a metastable set of observations of traffic densities at various junctions(modeled by nodes in the graph) of an operational traffic model is important to be able to begin this discussion. This is because we need a set of density observations as a starting point for the Three Regime(phase) traffic flow model to translate into capacities, which shall be used for our flow algorithm.

Following are the 3 expressions we will use to address traffic parameters, namely Traffic Density(k), Average Vehicle Speed(v) and Traffic Flow(q).

Free flow regime($k \leq 40$) :

$$v = 50 - 0.098k \tag{1}$$

Transistional flow regime($40 < k \leq 65$) :

$$v = 81.4 - 0.913k \tag{2}$$

Congested flow regime($65 < k$) :

$$v = 40 - 0.265k \tag{3}$$

where,

$$k = \frac{Vehicles}{Lengthof Road(mile)} \qquad q = \frac{Vehicles}{Time(hour)}$$

The expressions above are the relations under the Three Regime Model. [2]

The incremental nature of the traffic model is imparted by the continuously changing traffic densities. Perturbations in traffic density values, when large enough, can cause the traffic along the respective edge route to alter regimes. For scenarios employing continuous running of the flow algorithm, a close observation of these perturbations can play an important role in determining *when* the need for triggering the flow algorithm to reset flows on the new set of capacities arises. But, as a precursor to this, let us first establish the various necessary calculations to be made from the model which are used by the flow algorithm in traffic routing.

II. Design

Using the relations given by the Three Regime Model, we derive the relations for flow and maximize the expression to get capacities for the individual edge routes. Note that, these capacities are *average* values for the entire length of the edge route.

$$v = a - bk \qquad where a, b > 0 \tag{4}$$

where,

k(vehicles per mile)	a	b
$k \leq 40$	50	0.098
$40 < k \leq 65$	81.4	0.913
$65 < k$	40	0.265

TABLE I
VALUES OF A,B FOR GIVEN K

There is a universal relationship between q(flow), v(velocity) and k(density),

$$q = v \cdot k \tag{5}$$

In our case we are not so much concerned about velocity as we are about *average* speed of a vehicle. In fact, we are looking for capacity and average speed for each of the individual edges, through their dependence on traffic density values for those edges.

Using eq. (5), we can write flow as,

$$q = k(a - bk) = ak - bk^2 \tag{6}$$

As mentioned above, we maximize this expression for the flow, with respect to the traffic density. This maximum flow along any edge, we treat as the capacity for the particular edge. These capacity values changes as the traffic densities change over time, but the nature of change is a lot rarer. What that means is, for the capacity of an edge to alter, the density of that edge must change by a magnitude enough to pull it out of its current regime.

$$\frac{dq}{dk} = 0 \tag{7}$$

$$\Rightarrow \quad \frac{d(ak - bk^2)}{dk} = 0 \quad \Rightarrow \quad a - 2bk = 0$$
$$\Rightarrow \quad k = \frac{a}{2b} \tag{8}$$

We must check whether at this density(eqn. 8), we get maxima or minima.

For maxima,

$$\frac{d^2q}{dk^2} < 0$$

$$\Rightarrow \quad \frac{d^2(ak - bk^2)}{dk^2} \quad \Rightarrow \quad \frac{d(a - 2bk)}{dk} \tag{9}$$

$$\Rightarrow \quad -2b$$

Since b is a positive constant, $-2b$ yields a negative outcome for the double differential. Therefore, at the density in eqn. (8), we get an upper bound on the flow in the edge. We calculate this capacity in terms of the positive constants a and b, as,

$$\Rightarrow \quad q_{max} = a\Big(\frac{a}{2b}\Big) - b(\frac{a}{2b})^2 \quad \Rightarrow \quad a(\frac{a}{2b}) - b(\frac{a^2}{4b^2}) \tag{10}$$

$$\Rightarrow \quad q_{max} = \frac{a^2}{4b}$$

Tabulating this quantity for the 3 regimes :

Regime	Traffic Density(vehicles per mile)	constants a,b	Capacity(vehicles per hour)
Free Flow	$k \leq 40$	50 , 0.098	6377.55
Transistional Flow	$40 < k \leq 65$	81.4 , 0.913	1814.34
Congested Flow	$65 < k$	40 , 0.265	1509.43

TABLE II
EDGE CAPACITIES FOR 3 REGIME MODEL

III. METHODS

A networks traffic count is the data collected by traffic count devices like Automatic Traffic Recorders(ATRs) or off-road technologies using infrared beams to get speed and volume of traffic.

Modern techniques like 'Google Traffic' harness the GPS transmissions of locations of handheld devices. Knowing the collective speed of users along a stretch and the length of the stretch it is able to generate a live traffic map. [5] High Resolution Satellite Imaging could be a revolutionary measure in acquiring traffic count. The advantage with satellite imaging would be the amount of area it can span for measurement as against expensive single point measurements made from pressure sensors or electronic devices that are widely used today. Satellite imaging surveillance allows the acquisition of traffic density data for a large area, quantified in a stream of freeze-frames. We intend to device intelligent agents to make necessary deductions from the current stream of freeze-frames and offer us a fair estimate of the trend of traffic flow, in terms of weighted averages of time and distance values from a source point to a destination point($s - t$ traversal), for the timely varying (incremental) graph.

We will consider a sample graph to represent a city's sub-locality and induce timely changing traffic densities upon it to simulate a live scenario. In terms of networks, here we lend our focus to a *cluster* of the graph, which is the sub-locality in a city map. The vertices stand for the distinct end points of the routes in the sub-locality, and the edges indicate the routes themselves. There is one assumption that we consider here on out, which is, that when discussing the map of a sub-locality, we shall assume the routes in it are comparable in terms of distance. This stems from the fact that sub-localities are typically determined by clustering points based on their Euclidean distances. And since were talking about a sub-locality, its definition can also credibly be expressed in terms of the inter-connectivity of the vertices that make up the sub-locality cluster. In the bigger picture, a sub-locality is a region of relatively dense point locations which are more intensely inter-connected to one another than with the sundry point locations in the city map as a whole.

A. Algorithm

We settle with the flow algorithm as the workhorse to forward our study in interpreting a dynamic traffic situation. The rudiments of the flow algorithm lie in the Ford Fulkerson approach. The idea of the Ford Fulkerson method is to compute maximum flow between a source and a sink for a given network using augmenting paths. The Edmonds Karp algorithm, in turn, defines a criterion of picking augmenting paths from the graph at any stage of the flow algorithm. It picks the *smallest* augmenting path *in terms of the number of edges* at any stage and increments flow on it.

However, in our case, where the graph calculations are of dual behaviour depending upon their being based on the edge weights modeled by distances or by traffic densities, we need an adaptation of the Edmonds Karp algorithm which can either take into account the length of the edges firsthand, or, attempt to find a plausible incorporation of both the distances and the traffic densities in order to pick the augmenting path. This second metric would be the time of traversal across the particular edge. The reason that time as a metric is able to incorporate the duality of the graph is that it divides the distance of the edge route by the average speed of traffic on that edge route, where average speed is a function of the traffic density on it as per the 3 Regime Model. Thus, both distance and traffic density are accounted for, with time as a metric for grading augmenting paths.

Before we progress into further discussion, we will fork our flow method into 2 approaches and tag them the 'distance-centric ' flow algorithm and the 'time-centric ' flow algorithm. Stating it more formally, for the distance-centric model, the subsequent augmenting paths are graded as per their lengths from source to destination and the highest graded path, which in the distance centric approach is simply the shortest length path, is chosen and is *saturated*. Saturation is, in keeping with the norm here, induction of as much flow into the source to destination path as the minimum capacity edge route on that entire path can sustain.

We choose to term our distance-centric model as 'adaptive' Edmonds Karp because, as opposed to the original Edmonds Karp, the graph we deal with is dual weighted. Thus, subsequent augmenting paths are attained by a weighted graph search with each edge representing a distance. The results may not concur with those obtained by simply choosing paths with the least number of edges, as is the case with the original Edmonds Karp. A similar weighted search is carried out in the time-centric model, only that each edge in the graph now represents the average time of traversal along that edge.

B. Re-invoking Flow Algorithm

The flow algorithm is running on data fed through satellite images. Depending on the rate of incoming stream of snapshots, we could have a fair observation of how the traffic is altering in various parts of the map. However, each time a perturbation is recorded at any edge in the sub-locality's image stream, re-invoking the flow algorithm would be impractical. A perturbation means the change of traffic density across an edge by magnitude big enough for its regime to alter. So, we would be left monitoring for any single edge route to have changed regimes following which we would need to reset the flow by running the adaptive Edmonds-Karp.

In such circumstances where the flow needs to be reset timely, like in our study where the re-invoking helps reshuffle the optimal paths across an incremental graph, we assert a two step inspection which can potentially help prevent the re-invoking of flow even though the capacities might have altered. We shall first quickly acquaint ourselves with residual graphs, and 'bottlenecks '.

During the course of the flow algorithm's implementation, it is standard practice to maintain a *utility* graph, to keep track of the flow induced thus far and the prospective augmenting paths to use subsequently. At the time the algorithm has established a blocking flow we have a disconnected graph, with certain unused or partially flooded edge routes connected to either the source or the destination. The collection of such edge routes forms the residual graph. Bottlenecks is a term that we are using for those 'preliminary ' edges which are running full capacity by the end of flow establishment on the graph. Emphasis on preliminary because, if 2 successive edge routes of an augmenting path fall to zero residual capacity eventually, only the first one among them would be the bottleneck. The successive one, despite reaching nil residual capacity, would not qualify as a bottleneck since before the bottleneck action could have reached this ensuing edge farther away from the source, the precedent bottleneck will already have restrained the flow to its upper bound and the ensuing edge won't get the opportunity to produce a bottleneck effect on the already constrained flow. In fact bottlenecks are the same edges which form the minimum cut for such a source to destination flow graph.

The residual graph and the bottleneck serve as the most susceptible points to any perturbations in the graph. Therefore, by monitoring these across certain criterion could dictate precisely when the perturbations are strong enough to reshuffle flow. We establish a background for this in correlation with the traffic scenario.

Let us take a realistic case of a metropolitan city for a 3 hour period, from say, 6 am to 9 am. We will proceed with a qualitative analysis here to make the point of a stream of freeze-frame data more clear. Broadly speaking, in between 6am to 9am, the normal office reporting times, there would be a relatively large change in the traffic density over certain sub-localities of the city map, like the office compound parks, commerce centers of the city etc. After 9am, approximately, the traffic situation will again become somewhat more stable than than the comparatively radical change it just underwent. We are interested primarily in how the traffic changes its regimes. A radical change in traffic situation therefore, implies rapid changing across traffic regimes. A steadier state means the traffic stays in its current regime for an extended period. One could interpret this simply as 'rate of change of traffic regimes '.

Suppose we have traffic data across a sub-locality now. In order to make predictions for optimal routes, we would need flow algorithm to run again and again, in synchronization with perturbation across any edge route as it shows in the data stream incoming. And the process would be carried out for all source to destination pairs! This involves a lot of computation which *can be* avoided.

The solution lies in analyzing the residual graph of the current flow pattern(which means the flow that has been set after the most recent run of the algorithm) and the *bottlenecks* of the sub-locality graph for any pair of $s-t$ nodes.

We push our claim that, for any particular source to destination pair in the sub-locality graph, the traffic perturbations would necessitate the flow re-calculation *only* when,
1. The freeze-frame traffic density captured at particular instant causes any of the bottlenecks to alter regimes.
2. The decremental difference between the previous capacity of an edge upon which the flow algorithm was last invoked, and its new capacity, is greater than the residual capacity of that edge. This residual capacity is the unused capacity of the particular edge in the final residual graph of the flow established so far, through the most recent run of the algorithm. It implies that **not all** perturbations in densities that could cause the regimes of their respective edge routes to alter, would require the flow to be re-set also.

These conditions offer relatively small amount of cutback on the number of flow re-invokes if we let the nature of increment be absolutely unconstrained, meaning thereby that the weights are allowed to change drastically over time. However, the real beauty of the relent proffered by these conditions comes forth when we work with more realistic cases that have *some* constraint on the incremental nature of weights over time. We will bring this piece of discussion up in the result section once again.

Stating the 2 conditions mathematically, with the following denotations :

1. j denotes edge j,
2. b_j denotes the edge j which is also a bottleneck currently,
3. $k_{b_j}^{(t_i)}$ denotes the traffic density(vehicles per mile) on the bottleneck edge j at time instant t_i,
4. $\Delta_{b_j}^{t_i}$ denotes the amount of surplus or deficit in the traffic density of bottleneck edge j at time instant t_i in order for it to change its regime
5. $C_{r_j}^{t_i}$ is the residual capacity of edge j at time t_i.

Condition 1

$$\left|k_{b_j}^{(t_{i+1})} - k_{b_j}^{(t_i)}\right| > \Delta_{b_j}^{t_i}, \quad where \;\; k_{b_j} + \Delta \geq 0; \tag{11}$$

Condition 2

$$k_j^{(t_i)} - k_j^{(t_{i+1})} > C_{r_j}^{t_i} \tag{12}$$

Note, that if there is a change in the bottleneck densities such that their regime changes at all, then the difference between the new and the previous capacity anyway exceeds the bottleneck's zero residual capacity, and the flow needs to be reset. To reiterate the fact, this is because, for bottlenecks the residual capacity is nil(Converse is not true, as we have discussed before as to why only the 'preliminary' zero residual capacity edges make the bottlenecks!); and residual capacity being zero means any change in capacity of this bottleneck edge will either shrink the bottleneck further or open up the bottleneck, which would impact the flow on the network. Hence any change in regime of the bottleneck will be large enough to call for a flow reset, beyond which we needn't check for the second statement anymore. On the other hand it is possible for the regimes of the *generic* edges(those that are not bottlenecks currently) to reduce and yet not cause the flow to reset, since the decrement in their capacities is not *big enough*. Note also, that in case of generic edges, it is the 'decremental' nature of change that is of significance for the 2^{nd} condition which handles the generic edge perturbations. Which is to say that, whereas generic edges would only be sensitive to the *magnitude* of decremental perturbations of *capacity* values, even an incremental regime altering *capacity* perturbation for the bottlenecks would lead to flow reset. To sum up, first the bottlenecks should be checked for regime change. If they are in place(no regime change), only then must we concentrate on the extent of decremental capacity changes for all the other edges in comparison to their residual graph capacities.

1) Pseudo Code: **Definitions :** newSnapShot() returns the traffic at the present time. getCapacities() takes in the densities and returns the capacities as described in table II. Sort() returns the list of paths sorted by either length or time, depending upon the approach. The functions BNChangedRegime(), BN short for bottleneck; and excessCapacityChange() return true if the conditions described in equations (4) and (5), respectively, have been met, and the flow has to be re-invoked. IncomingTrafficFlow() gives the influx of traffic at the source in the present snapshot. Given below are some denotations for the expressions used in the pseudo-code.

$D[1,..,n,1...n]$: Density Matrix
$L[1,..,n,1...n]$: Length Matrix
$R[1,..,n,1...n]$: Residual Graph Matrix
$F[1,2,.....p]$: List of Maximum Flows in p graded paths
BN : List of bottleneck edges from the most recent run of the flow algorithm
$V[1,2....x]$: Set of visited points from source in residual graph

Algorithm 1 Adaptive Edmonds Karp Algorithm

Initializations
$s \leftarrow Source$;
$t \leftarrow Destination$;
$D \leftarrow newSnapShot()$;
$P \leftarrow DFS(s,t,L)$;
$P \leftarrow Sort(P)$;
$C \leftarrow getCapacities(D)$;
$CX \leftarrow C$;
$R, F \leftarrow EdmondsKarp(C,P)$
$B \leftarrow getBottleNecks(R)$

while (1) **do**
$D \leftarrow newSnapShot()$
$C \leftarrow getCapacities(D)$
$I \leftarrow IncomingTrafficFlow()$
$condition1 \leftarrow BNChangedRegime(B,C,CX,R)$
$condition2 \leftarrow excessCapacityChange(R,C,CX)$

if $(condition1 == true \;\; OR \;\; condition2 == true)$ **then**
$P \leftarrow Sort(P)$ (required only in the time-centric approach)
$R, F \leftarrow EdmondsKarp(C,P)$
$B \leftarrow getBottleNecks(R)$
$CX \leftarrow C$

end if
$z \leftarrow 0$

while $(I > 0 \;\; AND \;\; z < total\_paths)$ **do**
Direct F[z] amount of Flow through path P[z]
$I \leftarrow I - F[z]$
$z \leftarrow z + 1$

end while
end while

Algorithm 2 FUNCTION : EdmondsKarp(C, P)

```
R ←C
for each path in P do
maxFlow ← infinity
for for each edge (u,v) in path do
  if R(u, v) < maxFlow then
    maxFlow ← R(u, v)
  end if
end for
maxFlow ← capacity of the minimum capacity edge in
path in R

for for each edge (u,v) in path do
  R[u, v] ← R[u, v] − maxFlow
end for
F.addToList(maxFlow)
end for
return R, F
```

Algorithm 3 FUNCTION : getBottlenecks(R)

```
V ← DFS(R)
for each n in V do
for each edge (u,v) from n do
  if (R[u, v] = 0) then
    BN.addToList(u,v)
  end if
end for
end for
return BN
```

2) Complexity Analysis: In the distance-centric model, the paths can be graded in order of preference in $O(nlogn)$ time through sorting by individual path lengths . This is a one-time activity for a given sub-locality and does not need to be recomputed unless a new road is constructed or an existing road made inaccessible to the traffic. For the time-centric model, however, estimated time across each edge has to be computed with the changing traffic densities. Thus, the worst-case complexity of grading paths in the time-centric approach, assuming a dense graph, comes out to be $O(n^2)$ and this process has to be repeated each time the flow algorithm is invoked.

Each time the algorithm increments flow on an augmenting path, at least one edge becomes saturated. This in turn implies that after every stage of incrementing flow on an augmenting path, the length from source to sink inadvertently increases and hence is monotonic in nature. This goes on until a 'blocking flow' has been generated, which is to say that no source to destination path can capacitate any more flow whatsoever(i.e. no augmenting paths remain). Behaviorally, the graph is now disconnected into 2 components, one component connected with the source and the other with the destination.

For the 6 am to 9 am traffic situation cited previously, very roughly speaking, in sub-locality and all source to destination pair paths, the drastic altering of the bottlenecks might occur during the busy hour(say 8.30 am to 9.30 am), after which the traffic would stabilize relatively and the new bottlenecks would sit more stably in their new regimes. Which means, the flow algorithm would not need to re-invoke as rampantly through the less busy hours of the day! The second condition in our assertion, where we talk about the residual capacity of an edge providing the upper bound for the amount of capacity change in an edge, is in keeping with the traffic which may *arise from within the graph.*

Let us make this idea more concrete with a simple example. A residential locality's traffic patterns would involve the 2 broad phases. A 9am to 8pm phase of a regular weekday when the outflow of the residential locality is greater than the influx; and the remnant phase where the influx is greater than the outflow. This net change for the residential locality can be imparted to the people who live in this locality, and leave for work in the daytime phase(outflow $>$ influx); and return home in the other phase(influx $>$ outflow). It is fair enough to speak equivalently by saying that the locality has the property to *generate* traffic during the daytime phase, and *absorb* traffic during the night phase. Let us call this 'absorbed' traffic which can be 'regenerated' later into the locality's graph, the dormant traffic. So after the dormant traffic is generated into the graph in the daytime phase, it turns into active traffic; much of which would in turn become dormant in the commercial localities for the daytime phase of a working day.
The second statement, where we use the residual graph capacities as a monitor for all edges except the bottlenecks, mainly deals with this traffic which changes from dormant to active or active to dormant on any edge or set of edges. Inherently, through this form of traffic, the sub-locality map has the capability of *generating* traffic of its own, and also *absorbing* the traffic prevalent upon it. Clearly, this sort of traffic is not affected by the influx from source, or outflow through destination, and could arbitrarily change the capacities of a route. This is where the residual capacities help in maintaining check. Needless to say, if we could discount any such traffic absorption or generation from a locality, we could make do with only the 1^{st} statement for the bottlenecks, since in that case, any change in the route capacity would be due to change in the influx or out flow and such a change is sensed by the bottlenecks anyway.

IV. Implementation

To evaluate the two different approaches we have tried to simulate a real-time traffic scenario in the sub-locality represented by the graph in figure 1.

All edge routes are assigned initial random traffic densities. With the help of the 3 regime model the capacities for the route are computed, corresponding to this current traffic density situation. As per the density - average speed relations we introduced before in equations (1), (2) and (3); for any of the 3 regimes we can find average edge route capacities as shown in equation (10).

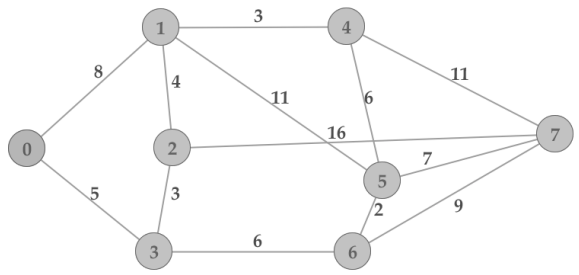

Fig. 1. Sub-Locality Graph with edges representing point-to-point routes, and edge weights representing lengths of the respective routes

In a similar manner the expected speeds are computed, using equation (4). We first invoked the flow algorithm for both the distance and time-centric approach to get the s-t paths in order of preference for augmenting flows, and correspondingly the appropriate flows for each path as given by the flow algorithm. We then run an iteration to generate a large number of successive traffic density snapshots with randomly changing densities along each edge. This random change we constrain to a small positive or negative bound. At each iteration a check is performed to see if the conditions described in equations (4) and (5) are met and if the flow algorithm has to be invoked again. A random influx of traffic, we refer to this as 'test traffic', is taken at the source and is directed along routes corresponding to the output of the flow algorithm. The test traffic is one test lot of vehicles that we are enforcing into the graph and trying to steer all of it from source s to destination t. This is a step for convenience of calculation afterwards, since, in actual traffic scenarios any individual vehicle among this lot could be headed its own separate way from a different source to an altogether different destination. In such a case, the algorithm would be made to run for the particular s-t pair that the individual vehicle wants to traverse. What would be common to all vehicles headed their separate ways is the particular sub-locality and its current traffic situation as captured by the most recent satellite image. Therefore, in algorithmic terms, each vehicle would be running the model to eventually impart flows on the same capacity graph for different source to destination pairs.

Getting back to our test traffic, we subject this common lot of enforced traffic to the scenario as given by the snapshot and employ our approach for optimal path search with each preferential route receiving its maximum amount of flow from the test traffic lot, before we choose the next preferable route for the remaining test traffic and so on. To evaluate the two approaches we use the average distance travelled by the individual vehicles among the test traffic and the average speed of these individual vehicles estimated at the *start* of the journey. Evaluating the speed at the start of the journey is cause for discomfort for any approach, since the value of speed at the beginning is hardly intuitive about how the journey will fare at a later point. Let us first address this before we move forward with the implementation.

To clear the idea, we must visualize the case of real time traversal of a vehicle. From some source *s*, the driver runs the model for particular destination *t*, and is offered the prospective paths graded as per distance or time. Through the course of his/her journey, it is possible that the traffic densities change in a manner that the flow needs to reset. This implies that the optimal paths leading to *t*, may change and may not anymore include the passage he/she is on currently. What such a flow reset is supposed to commit to the model to do is, that every time the flow does reset for s-t, a new flow must be computed for the traveller in the midst of his/her journey, taking the *very next node as source* s' and the original destination *t*. In such a manner the traveller can be efficiently re-routed. Also, at this reset, an average speed will be re-generated.

Thus, every time a motorist is re-routed, at the time of flow reset *from his most recent source to destination pair*, an average speed is furnished. This average speed value is meant to hold plausibly until another major perturbation in the traffic densities causes the flow to reset and the motorist to be re-routed during his travel.

For the purpose of examining results, we calculate the weighted-averages of distance, time of traversal and average speed for the test traffic for a fixed, single source to destination. Here, since we are concerned with 'weighted averages' of the test traffic as a whole in order to bring out the performance of the algorithm, we do not simulate the individual motorist's journey and the re-routing and speed escalations or drops he/she faces during the travel. Finally, using the 2 conditions stated above, we keep count of how many times the flow algorithm is invoked in each case since re-evaluating the paths is a computationally expensive process and should be minimized. We repeat this exercise multiple times to draw out the above metrics and, possibly, notice some trends as to which approach works better in each front. We briefly present the working of the 2 flow-reset determining conditions through the following demonstration.

Now that we have a grasp on how to calculate capacities from a graph for which the satellite image data gives traffic densities, we shall for sake of ease of demonstration talk only in terms of capacities of the edges of the sub-locality graph. Let us assume the capacities of the edges of the graph at a particular time is :

In the figure 2, we show rounded off capacities to allow better clarity in presentation. Here after the flow is run for source vertex 0, destination vertex 7; and min-cut is obtained, we get edges $0 \rightarrow 1$ and $0 \rightarrow 3$ to be the bottlenecks. Which means for any perturbation arising due to external factors through the source vertex 0 of the graph, the bottlenecks will be the most sensitive zones. To monitor any interior perturbation causing change of capacities, we will use the residual capacities of the other edges of the graph. This has been projected in the figure 3 given below.

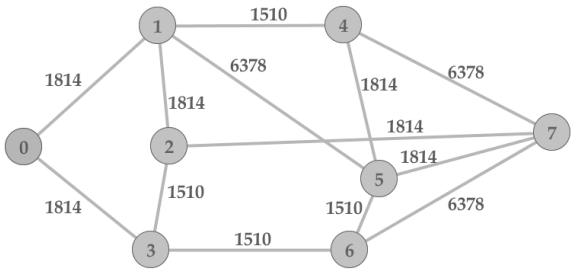

Fig. 2. Sub-Locality Graph with edges weights representing capacities(vehicles/mile) of the respective point to point edge routes

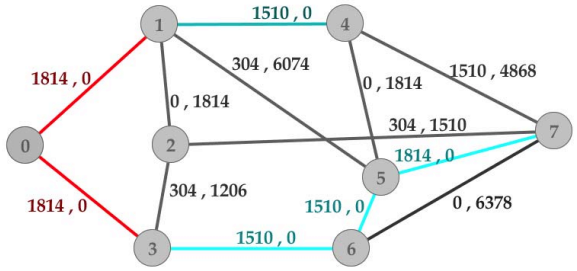

Fig. 3. Sub-Locality Graph with edges weights representing flow induced(vehicles/mile) along with the residual capacities(vehicles/mile)

Here, the edge weights depict the flow and the residual capacities at the end of the flow algorithm's implementation for one set of freeze frame data. The red edges are the current bottlenecks(current implying for this particular freeze frame data), and the cyan edges represent the zero residual capacity edges which are *not* the bottlenecks. Going by the 2 conditions, any change of traffic densities in the forthcoming freeze-frames that causes the red edge capacities to alter even slightly(thereby causing those edges to alter regimes since they are bottlenecks), incrementally(capacity increases) or in a decremental fashion(capacity decreases) **will** call for a flow reset. On the other hand, for any of the other edge to cause a flow reset, say edge 2→3, the subsequent freeze-frame density for this edge should be such that its new capacity is *lesser* than 1510(its current capacity) by at least 1207(its current residual capacity + 1). Putting it tidily, if the capacity from the subsequent freeze-frame is any lesser than 304, the flow will need to reset. Note, this edge not being a bottleneck qualifies only for the decremental check, i.e. an increase in the capacity of this edge will not require a flow reset. We halt this section by stating the difference between the cyan edges and the red bottleneck edges. For the red edges the regime change required to bring about a flow reset can be incremental or decremental. Meanwhile for the blue edges, only decremental change of regimes can cause the flow to reset.

V. Results

The adaptive-Edmond's Karp is employed for 1 million test cases, which basically means the traffic densities on the sub-locality being made to change 1 million times, randomly. The magnitude of this change will have an upper bound since traffic generation at any source, or traffic absorption at any sink can not be infinite. Also there will exist an upper bound for the *rate* of change of density values. Depending upon the frame rate of the stream of freeze frames obtained and the overhead of the feature extraction and other techniques used on each frame to estimate traffic density values; and experimental data for how radically can traffic demographic change over time, we can determine the constrain on the increment or decrement of edge densities.

We observe the effect of this continuously changing scenario on the test traffic from our chosen source(here it is node 0) to the chosen destination(here it is node 7). The tabulated data below gives value of the distance, time and speed weighted-averaged for the entire lot of the test traffic that was induced across the incrementally changing sub-locality graph.

Augmenting Criterion	Average Distance	Average Time	Average Speed	Algorithm Re-invoked
min. Distance	22.74841	1.10506	20.58572	265
min. Time	25.809432	0.89769	28.75098	257

TABLE III
ADAPTIVE EDMONDS KARP RUN 1 FOR S=1, T=6

Augmenting Criterion	Average Distance	Average Time	Average Speed	Algorithm Re-invoked
min. Distance	21.76136	1.14345	19.03127	168
min. Time	26.59955	0.88812	29.95033	186

TABLE IV
ADAPTIVE EDMONDS KARP RUN 2 FOR S=1, T=6

Augmenting Criterion	Average Distance	Average Time	Average Speed	Algorithm Re-invoked
min. Distance	21.94234	1.22269	17.94598	41
min. Time	26.61533	0.91931	28.95151	43

TABLE V
ADAPTIVE EDMONDS KARP RUN 3 FOR S=1, T=6

Note that, the efficacy of the 2 conditions which monitor the incremental nature of the graph to ensure when the flow must reset attests our affirmation. The 1,000,000 increments in the graph which would potentially need 1 million flow resets, has been tremendously reduced to the order of a few hundreds! This is to say that we saved upon the huge proportion of the potential re-routing and thereby saved enormously *repetitive* computation that the flow algorithm would otherwise employ. We can quantify the benefit we achieved through this by time-framing our algorithm and pitting it against a case where the flow would need to reset each time the traffic densities were even slightly perturbed.

Given below is the comparison between the 2 approaches, for tries ranging from 10,000 freeze frames up to a 100,000.

We monitor how our conditioned flow re-invoke model performs against the standard.

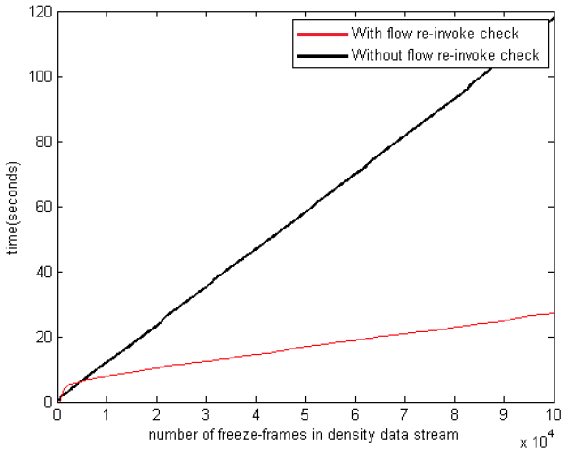

Fig. 4. Plot depicting comparison of the conditioned approach against the standard approach with respect to time of implementation for 100,000 test case frames

We must bear in mind, that in our approach we have stuck an upper bound to the magnitude of perturbation that occurs in the traffic density values for the edges. The number of flow re-invokes gets higher as the upper bound on the perturbation loosens. In the worst case scenario, where perturbations can range arbitrarily without any bound, the flow is re-invoked anywhere between 84 to 88% of the times. Again, a worst case scenario implies we are loosening *all* bounds for the perturbations across any edge of the graph. For a traffic scenario, that perturbation could extend anywhere between 1 to 130 vehicles/mile in terms of traffic density perturbations, *within a single freeze frame*. So, for a modest 50 frame per second streaming, we might be looking at an accumulation or deduction of up to 130 vehicles on a stretch of 1 mile within a little more than $\frac{1}{50}$th of a second, give or take(since we're also accounting for the time overhead of the computer vision techniques that help get the density data from the images). This type of vehicle demographic change is unrealistic, and so we appreciate the use of an appropriate upper limit to the perturbations. Also, note from the figure 5, that at very few number of runs of the flow algorithm, the conditioned approach does not perform convincingly compared to the standard flow reset approach, that re-evaluates the flow for every incoming frame, without exception. This again is because at such a small number of frames the cutback offered by the conditioned approach is not able to formidably outperform the overhead it suffers due to the depth first search for monitoring the bottlenecks.

Through this entire exercise, our construction was for optimal re-routing in an incremental traffic scenario, but we must explicitly state the relevance and applicability of the result at the heart of this study. There could be a variety of problems requiring continuous flow calculation time and again, as the capacities of the network change over time. Having a fair idea about the general order of magnitude of change(perturbation) in the capacity values of the graph makes the application of the conditional flow re-invoke exponentially better. Thus, we're able to salvage the practicality of *repeatedly* running the $O(VE^2)$ flow algorithm over an extensive graph and save precious time.

VI. Discussion

There are some parts of the thesis above that we would like to address. We mentioned previously the importance of adhering to the boundedness of the perturbation value. Through demonstration we intend to make this point clearer.

In figure 4 we had used the perturbation to be upper bounded by 1 vehicle/mile, meaning thereby, if the stream is at 50 frames per second, then in a little more than $\frac{1}{50}$th of a second, one mile of a stretch on average gains or loses 1 vehicle. Here, in successive freeze frames we delimited the magnitude of change of the traffic density for every edge to not go beyond 1, by increment or decrement. We will now project the comparisons of more loosely bounded test cases on our conditioned flow algorithm against the standard repeated flow evaluator and observe how we lose upon the efficiency of our approach in terms of time. The important thing to keep in mind is that even in the worst case, the *number of re-invokes* is still fewer than the number of times the standard algorithm re-establishes flow(viz. equal to the number of frames). The factor that raises the time consumption, despite the number of flow re-invokes being fewer is the fact that for our conditions to operate, we need to keep track of the bottlenecks each time a new flow had been established. This means an added depth first search must be performed at the time of the flow re-establishment in order to track the minimum cut of the flow graph, and hence its bottlenecks. Through a strong constraint(a precise constraint, that is) the time consumption of the depth first search is far more over-compensated by the huge cutback we have accomplished on the number of re-invokes.

A. Further Work

We have explored the use of flow algorithms which exercise toward *maximizing* flows in the given network and taken that as an empirical basis to judge traffic optimal paths from any source to destination in an incremental graph. We analyzed the changing traffic scenarios including the external factors of a sub-locality, such as radical increase in the inflow or outflow of traffic from the sub-locality, and, we accounted for internal factors such as 'dormant traffic'.

The flow algorithm in turn can be modified in context of its selection of augmenting paths. Our augment occurs with the help of the 2 metrics that we have discussed. This decision making could further be explored by bringing in capacities to play a more frontal role than their effect on the time metric.

Another point which warrants a promising study is developing a criterion that *quantifies* the selection of suitable perturbation limits for a particular model, by taking into account the experimental data which the model runs on. In

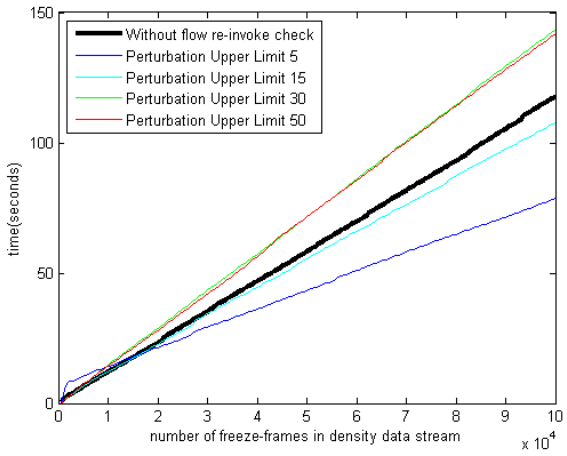

Fig. 5. Plot depicting comparison of the conditioned approach with various perturbation upper bounds against the standard approach with respect to time of implementation for 100,000 test case frames

our work we have only qualitatively explored how reactive our conditioned flow algorithm is to the boundedness of the perturbation constraint, through observation of time of implementation of the algorithm over a number of cases with varying values of perturbation upper limits.

ACKNOWLEDGMENT

We would first like to thank the university's academic administration for allowing us the chance to work on this thesis under good supervision. We also express boundless gratitude toward our family members for their motivation, time and again. Last but not least, we owe much to the patience of our friends who offered an interactive audience to our arguments pertaining to this work and opened up many new avenues of discussions to make this study more concrete.

REFERENCES

[1] B. Kerner, *Complexity of Synchronized Flow and Related Problems for Basic Assumptions of Traffic Flow Theories*, vol. 01 (41):35-76 Networks and Spatial Economics, 2001.

[2] Tom V. Mathew, *Traffic Stream Models*, 01(41):3.1-3.11 Transportation Systems Engineering, 2014.

[3] Siri Oyen Larsen and Hans Koren and Rune Solberg, *Traffic Monitoring using Very High Resolution Satellite Imagery*, Photogrametric Engineering and Remote Sensing, 2009.

[4] Jack Edmonds and Richard Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, 19(2):248-264 Journal of the Association for Computing Machinery, 1972.

[5] K. Subramanian, *Now, Apps for Live Traffic Feed*, The Hindu, 2014.

[6] Ulrich Laube and Markus Nebel, *Maximum Likelihood Analysis of the Ford-Fulkerson Method on Special Graphs*, , 2013.

[7] Jeffrey Erman and Martin Arlitt and Anirban Mahanti, *Traffic Classification Using Clustering Algorithms*, 281-286 , SIGCOMM Conference, 2006.