The End of Rush Hours: A Schedule-Driven Approach to Demand Shifting

Abstract

In this paper, we propose an approach to mitigate rush hour congestion through demand shifting. We consider a setting where drivers have flexibility in the times that they can arrive and depart from the office, and all drivers must pass through a congestion bottleneck (e.g., a tunnel). The goal is to produce a departure time for each driver that minimizes congestion at the bottleneck. We first formulate a base constraint satisfaction scheduling problem with two types of constraints: a set of departure windows, which specifies the temporal flexibility that each vehicle has in starting its commute (and timing its arrival at the bottleneck), and capacity constraints, which specify an upper bound on the number of vehicles that can feasibly arrive at the congestion bottleneck simultaneously. Since the level of congestion at the bottleneck corresponds directly to the capacity constraints, we seek to find a feasible schedule that minimizes its value. Two iterative algorithms are proposed for solving this problem and analyzed using real-world map data of Pittsburgh, Pennsylvania. Compared to a baseline case where each vehicle starts at its earliest time, these algorithms are found to reduce mean travel time by $15 - 24\%$. We also explore the variation in mean travel time under different assumed penetration rates (i.e., percentages of vehicles using the scheduling service). The improvement is still found to be substantial with a 40% penetration rate.

Introduction

Traffic congestion in urban areas is a serious problem, resulting in significant economical cost for drivers through wasted time and fuel, and environmental cost due to increased vehicle emissions. The major cause of traffic congestion is limited roadway capacity, which is highlighted during peak rush hours when the number of vehicles on the road exceeds the capacity of "bottleneck" road segments. One intuitive approach to address the congestion problem is thus to add infrastructure to increase capacity. However, traffic and road capacity analysis studies have shown that widening and building more roads actually creates more traffic (Downs 1962; Duranton and Turner 2011). Furthermore, increasing capacity can be expensive, and in some cases is simply infeasible due to geographic conditions or land use issues. For all these reasons, adding road network capacity cannot provide a viable total solution to traffic congestion and must be coupled with other methods.

Due to the large number of vehicles traveling during peak rush hours, roadways are especially susceptible to congestion if all vehicles travel during the same interval, as might naturally be the case for typical commuters. Over the past decade, major cities are increasingly turning to demand management approach as a way of regulating the number of vehicles on the road and relieving traffic congestion (Meyer 1999; Pluntke and Prabhakar 2013). However, existing approaches to demand management such as fee charging or road restriction policies are often not well aligned with public interests and resisted due to perceived unfairness to various segments of the driving community. A more driver-friendly method of regulating the number of vehicles on the road is needed.

In this paper, a new schedule-driven demand shifting scheme is proposed as a mean of relieving traffic congestion. This scheme explores the potential for reducing traffic volume by scheduling vehicles according to the collective demands of participating drivers. In brief, each driver is assumed to have some temporal flexibility as to when they leave for work and return home. The proposed scheme gathers this information, together with geographical information about the commute locations of each driver, and produces a schedule that meets these constraints while distributing travel intervals to the extend possible. The schedule provides each participating driver with an indication of when the best time to depart is, using the diversity in various driver's demand requirements to achieve a reduction in peak hour traffic volume.

To demonstrate the potential of schedule-driven demand shifting scheme, the problem is formulated as a constraint satisfaction scheduling problem (Minton et al. 1992; Cheng and Smith 1997) with two types of constraints, a set of departure window constraints and a set of capacity constraints. The set of window constraints specifies the temporal flexibility that vehicles have in starting their commute. The capacity constraint for each time slot specifies an upper bound of the number of vehicles that can arrive at the congestion bottleneck simultaneously. Since the level of congestion directly corresponds to the capacity constraints, the upper bound should be minimized to achieve reduction in traffic

congestion. In order to solve this optimization problem, we develop and implement two iterative scheduling algorithms. The first developed algorithm is modified from min-conflicts local search algorithm (Minton et al. 1992); the second is an iterative version of squeaky wheel optimization (SWO) (Joslin and Clements 1999). In the experiments, we consider Fort Pitt tunnel of Pittsburgh as our bottleneck. As a default baseline strategy, we assume that drivers depart at their earliest departure time (to maximize their chances of arriving on time). A performance comparison was carried out with the schedules produced by both iterative procedures, and relevant performance metrics, e.g., mean travel time or computational time, were collected for each test condition. Across all comparisons, the two algorithms achieve significant performance improvement, ranging from $15 - 24\%$ in travel time reduction. In addition, the scheme is shown to improve traffic congestion under more limited assumptions as to the percentage of vehicles that are using the scheduling service.

The remainder of the paper is organized as follows. First, the setting of the problem is described and formulated as a scheduling problem. We then introduce related approaches in the research community and the existing regulating methods of relieving traffic congestion repectively. Two iterative scheduling algorithms are then presented. Finally, the simulation results and evaluation of schedule-driven demand shift scheme are discussed, followed by conclusions.

The Demand Shifting Traffic Scheduling Problem

We are given a set of vehicles $v \in V$. Each vehicle v has a specific route connecting starting location and final destination. When vehicles are traveling from the starting location to the destination during rush hours in the morning and in reverse direction in the evening, most vehicles will pass through several important road sections and lead to huge traffic volume. We call these road sections bottlenecks (Helly 1900). For example, the Fort Pitt tunnel is the most important road section connecting downtown and the suburban South Hills area outside of Pittsburgh PA. When the amount of traffic exceeds the capacity of the bottleneck, traffic congestion will emerge and delay will be incurred. In the following sections, we consider the case of a single bottleneck for simplicity.

Each vehicle v is assumed to have some flexibility as to departure time. By exploiting the collective flexibility of all drivers, vehicles can be directed to arrive at the bottleneck at different times and alleviate traffic congestion without constructing new infrastructure. In contrast to the concentrated high traffic volume that would pour into the bottleneck road segment otherwise during short peak hours, scheduling vehicles based on drivers' demands spreads the commute patterns of vehicles across a longer duration and vehicles encounter fewer vehicles competing for the same road capacity.

Let $N = |V|$ represent the number of vehicles that must traverse the bottleneck road segment (e.g., the Fort Pitt tunnel) during rush hour. Each vehicle v has:

- A starting location *start(v)*, and a final destination *dest(v)*.
- A departure time window $W_s(v) = [e_s(v), l_s(v)]$ in which v must depart to its *dest(v)*.
- A time shift *stob(v)* defines the estimated time to travel from *start(v)* to the common *bottleneck*
- A shifted version of time window $W(v) = [e(v), l(v)] = [e_s(v) + \textit{stob(v)}, l_s(v) + \textit{stob(v)}]$ in which v will arrive at the bottleneck segment.

For simplification, the rush hour period T_{rush} containing all $W(v)$ is discretized and indexed into K equivalent time units ΔT. We then round down the $W(v)$ to the nearest time index within T_{rush}.

The goal is to form a schedule, S_s, which specifies when each v should depart within its specified window $W(v)$ to avoid traffic congestion, where $S_s[v_1, \ldots v_N] = \{s_s(v_1), \ldots, s_s(v_N)\}$ is a set of departure times of all vehicles. We define the number of vehicles that arrive at the bottleneck as the traffic volume. The objective is then to reduce the traffic volume at the bottleneck per unit time ΔT while satisfying each vehicle's departure window constraint. In fact, it is noted that finding S_s is equivalent to finding the arrival time at the bottleneck, which is $S[v_1, \ldots v_N] = \{s(v_1), \ldots s(v_N)\} = \{s_s(v_1) + \textit{stob(}v_1\textit{)}, \ldots, s_s(v_N) + \textit{stob(}v_N\textit{)}\}$, when we are given $\textit{time\_to\_bottleneck}[v_1, \ldots v_N] = \{\textit{stob(}v_1\textit{)}, \ldots, \textit{stob(}v_N\textit{)}\}$. In the following description, we will use S to formulate the problem, and S_s can be easily derived after shifting S in a backward manner. A valid (feasible) schedule must have the following properties:

- A valid **arrival time**. The domain of $s(v)$ should be the discretized rush hours $T_{rush} = \{1, \ldots, K\}$.
- A **window constraint** for each v, which specifies when the vehicle is allowed to arrive at the bottleneck based on its $W(v)$. We assume the consecutive time slot indexes covered by $W(v)$ are the integers, $e(v), \ldots, l(v)$, and then the following constraints must be satisfied:

$$e(v_i) \leq s(v_i) \leq l(v_i), i = 1, \ldots, N. \quad (1)$$

- A **capacity constraint** for each time slot $k \in T_{rush}$,

$$\sum_{i=1}^{N} \mathbf{I}(s(v_i) = k) \leq c, \quad (2)$$

  where $\mathbf{I}()$ is the indicator function, and c is a upper bound of the number of vehicles that arrive at bottleneck simultaneously. The c can be determined by the free flow rate at the bottleneck.
- An **driver input constraint** for each v. The driver input constraint identifies invalid driver input data.

$$l(v) - v(v) \geq \Delta T, \quad (3)$$

  which means that for all vehicles the minimum time window duration should cover at least one time slot so as to ensure that the problem is solvable for each vehicle.

Given the above constraint satisfaction scheduling problem, the overall optimization problem is to minimize c. By minimizing c, we lower the number of vehicles that can be

feasibly scheduled to travel through the bottleneck segment simultaneously, and the fewer the number of vehicles that can arrive simultaneously, the shorter the delay in traversing the bottleneck segment.

The overall abstraction of our problem is depicted in Fig. 1, in which we have four vehicles with their preferred time window $W(v)$. The task is to specify the value of $s(v)$ so as to determine the exact departure time for each vehicle after being scheduled.

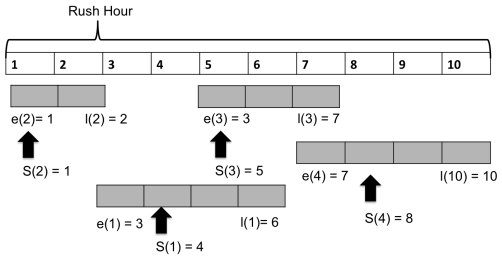

Figure 1: Four vehicles case for the demand shifting traffic scheduling problem

Related Approaches

The traffic scheduling problem defined in this work is a new problem due to its objective of reducing upper bound of capacity constraints. If the upper bound is fixed and without the variable departure times of each user, we can view the problem as an evacuation planning problem. In macroscopic evacuation planning, it models the movement of evacuees as flows in the evacuation graph. The evacuation planning is related to maximum dynamic network flow problem (MDFP) (Ford Jr and Fulkerson 1958; Burkard, Dlaska, and Klinz 1993). MDFP computes the maximum flow over one time period and then determine the paths on the network through flow decomposition. However, MDFP and related problems do not consider the variable departure times of each flow to compute maximum flow in the context of evacuation. (Even, Pillac, and Van Hentenryck 2015) introduced the concept of convergent evacuation plans, which assigns an evacuation routes to each residential zone. The authors defines a time-expanded graph (Köhler, Langkau, and Skutella 2002) to tackle temporal aspects of evacuation planning, which could correspond to the variable departure times of traffic scheduling problem. However, getting optimal solution of this problem is restrictive because of its scalability issues. Using it to solve traffic scheduling problem is even worse, because all drivers have a window, which increases the number of nodes in the graph by at least one order. In addition, the graph of the traffic scheduling problem is composed of few congested bottlenecks and a great amount of source nodes that represent the starting locations of drivers. According to commuting behavior, it is preferable to model it as a scheduling problem since the drivers usually intend to select main roads for commuting.

The traffic scheduling problems can be also viewed as a dynamic job shop scheduling problem (Ramasesh 1990) when the vehicles are jobs and the bottleneck is machine. The single machine problem usually assumes the capacity of the machine is constant at a finite value. However, the upper bound of road capacity that leads to free flow is difficult to determine since it depends on the chosen speed and the definition of congestion level. If the upper bound can be minimized and simultaneously fulfill the time requirement of each user, the travel time would be reduced as much as possible and also be able to incorporate more traffic uncertainty.

Developing control strategies to reduce traffic congestion in the research community has been discussed extensively. This includes traffic light control (Boillot, Midenet, and Pierrelée 2006), contraflow (Kim, Shekhar, and Min 2008) and rerouting of cars (Li, Mirchandani, and Borenstein 2009). Compared to those works, this paper considers a more realistic and simpler approach, which only takes user's demand into account without adding controlled infrastructures, to solve the traffic congestion. Moreover, this approach can be operated with other approaches and provide a synergy effect. To us, the traffic scheduling problem falls into a category of traffic regulation design.

Existing Solutions to Traffic Congestion

Since traffic congestion has been a substantial problem in our society, researchers from different fields such as traffic engineering and economics have been persistently working to eliminate the cost of congestion. Current approaches to congestion follow three basic paradigms:1) capacity addition, 2) road rationing ,and 3) congestion charging.

Capacity addition solves congestion from the supply side. Increasing the number of lanes and deploying new public transit services are two approaches of capacity addition in transportation networks. However, there are several challenges with increasing the capacity of road networks. For example, the fundamental law of expressway congestion (Downs 1962; Duranton and Turner 2011) claims that the traffic volume of an expressway increases with additional road capacity, and construction of new capacity does not reduce congestion for existing commuters in the long run.

From the demand side, road rationing and congestion charging (Liu, Yang, and Yin 2014; Han, Yang, and Wang 2010; Viegas 2001) are two other approaches for alleviating traffic congestion. For road rationing, certain types of vehicles are prohibited from being driven on certain days. Similarly, congestion charging means that vehicles are charged for the usage of the road. However, both approaches run into public resistance due to cost of deployment and unfair pricing. For example, New York City abandoned its congestion charging policy for Manhattan after encountering significant opposition from neighboring suburbs, who considered the scheme to be unfair.

The above demand-side congestion mitigation methods modify user behavior by disallowing or restraining drivers from on the road. On the other hand, the schedule-driven scheme we propose in this paper exploits each driver's commute flexibility to search for departure times that disperse congestion. Compared to other methods for demand management, the proposed scheme is more closely aligned with human behaviors and easily applied in cities due to

the development of mobile computing (Thiagarajan et al. 2009) and crowd sourcing applications(Pluntke and Prabhakar 2013).

Minimizing the Upper Bound by Revising Schedules

To relieve traffic congestion, the departure times of all drivers must be diverse. Therefore, we seek a feasible schedule that minimizes the upper bound of the capacity constraints. Through the schedule, the number of vehicles that arrive at the bottleneck simultaneously is decreased. In this section, we propose two iterative scheduling algorithms to achieve the goal. These algorithms are:

1. **Iterative Min-Conflicts**. The iterative version of min-conflicts local search starts by randomly choosing a schedule that fulfills all constraints. This schedule is then improved by reducing the upper bound c and feasibly reinserting those vehicles whose time assignments are now in conflict.
2. **Iterative Squeaky Wheel Optimization**. By applying a greedy schedule generator and reprioritizing vehicles based on the results, iterative SWO is able to form a schedule that minimizes not only the upper bound c but also the delay experienced by each driver. The schedule is then improved iteratively by reducing the upper bound until a feasible solution does not exist.

Iterative Min-Conflicts

The demand shifting traffic scheduling problem can be viewed as a large scale constraint satisfaction problem (CSP). Due to the large number of vehicles in real world cases, min-conflicts local search is a promising candidate as an efficient solution procedure. However, as mentioned in the previous section, the traffic at the bottleneck needs to be dispersed to the extent possible. Since min-conflicts local search needs a fixed capacity constraint to find a feasible solution, we define an extended iterative version that progressively lowers capacity constraint c each time a feasible solution is found.The algorithm consists of four parts. From highest to lowest level, these are:

Assignment Initialization Start with a complete assignment of S. The assignment is randomly chosen from the domain $T_{rush} = \{1, \ldots, K\}$ and has to satisfy the window constraint of each vehicle.

Value Selection With the initial assignment, those vehicles violating the capacity constraints are identified. `min_conflicts` randomly selects a v among the conflicted vehicles. The vehicle is then moved to a time slot that contains less vehicles. Namely, the arrival time $s(v)$ that minimizes the number of violated constraints is chosen. The process will be repeated until a conflict-free solution is found or a maximum number of iterations is reached.

Shrinking the Capacity Constraints Once a solution is returned, we reduce the upper bound c by a fixed amount δc, and identify a new set of conflicted vehicles as described in the middle and right sides of Fig. 2.

If we initially set the upper bound to a small value, it is possible that a feasible solution does not exist (or cannot be found in a fixed number of trials). Hence, we instead choose a larger initial upper bound and revise the returned solution at each iteration. If there is ultimately no feasible solutions under current value of capacity constraint c, min-conflicts will just return the solution in previous iteration.

Variable Ordering After reducing the upper bound c, the subroutine `order_variables` selects the new set of conflicted vehicles that have the largest time window. Those vehicles are then moved to other available time slots by `min_conflicts` according to the reduced capacity constraints. As the conflicted vehicles are redistributed, the distribution of $S(v)$ is becoming more diverse. As shown in Alg. 1, this two step process is repeated until either a conflict free solution is found or the maximum number of iterations is reached.

Algorithm 1 `iterative_minconflicts`$(S, c, \delta c)$: Form a schedule from an initial schedule S with initial upper bound c. δc is the decrement of capacity at each iteration

$C \leftarrow$ an array of size T_{rush} initialized to c
$R \leftarrow S, S \leftarrow \emptyset$
while $c > 0$ **do**
  $R \leftarrow$ `min_conflicts`(R, C)
  $S \leftarrow S \cup R$
  if $R \neq \emptyset$ **then**
    $c \leftarrow c - \delta c$
    $R' \leftarrow \emptyset$
    for $k \in T_{rush}$ **do**
      $r \leftarrow c - \delta c - \sum_{s \in R} \mathbf{I}(s = k)$
      $C(k) \leftarrow \max(r, 0)$
      if $r < 0$ **then**
        $R' \leftarrow R' \cup$ `order_variables`$(R, k, |r|)$
      end if
    end for
    $R \leftarrow R'$
    $S \leftarrow S \setminus R$
  else
    $S \leftarrow S \cup R'$
    return S
  end if
end while

Iterative Squeaky Wheel Optimization

For all commuters, we assume it is more desirable to leave office and home as early as possible,and hence specify an iterative version of SWO to minimize both the upper bound c of the capacity constraints and the delay experienced by each vehicle. Alg. 2 shows the iterative SWO algorithm. Generally, SWO consists of three parts: a *constructor*, an *analyzer*, and a *prioritizer*. In the constructor, a greedy algorithm is used to create a solution that respects the c, relaxing a driver's time window if necessary to get it into the schedule. The analyzer is then applied to find those vehicles

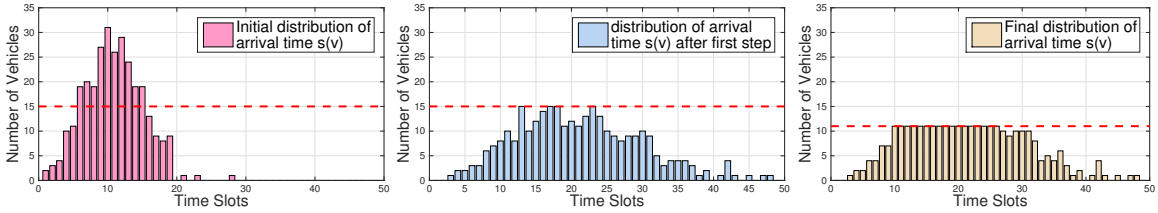

Figure 2: Solution evolution of the iterative min-conflicts algorithm during two iterations of reducing upper bound c. Left: original distribution of the departure times. Middle: after one iteration using original capacity constraints. Right: after two iterations using the reduced capacity constraints.

whose $s(v)$ are outside of the time windows. If $S(v)$ is improved, it is likely to reduce the number of violating v. The results of the analyzer are used to reprioritize the order in which drivers are scheduled by the constructor and the process repeats. This process terminates when either a conflict-free solution is found or a maximum number of iterations is reached. If a solution is found, c is reduced and the core SWO is reinvoked.

Algorithm 2 `iterative_swo`$(c, \delta c)$: Form a schedule with initial capacity upper bound c. δc is the decrement of capacity at each iteration

$score \leftarrow \infty$
while $c > 0$ **do**
  $P \leftarrow$ all vehicles sorted from the least window to the greatest window
  for $i = 1, \ldots$ Max **do**
    $S \leftarrow$ `greedy_insert`(P, c)
    $B, score \leftarrow$ `calculate_blame`(S)
    $P \leftarrow$ `prioritize_seq`(P, B)
    if $score = 0$ **then**
      $R \leftarrow S$, break
    end if
  end for
  if $score > 0$ **then**
    return R
  end if
  $c \leftarrow c - \delta c$
end while

Constructor: Greedy Insertion First, all vehicles are sorted to form a prioritized sequence according to the window size heuristic from least to greatest. Then, `greedy_insert` is called to sequentially insert v into the schedule S. Every v is inserted in the first free time slot found moving forward in time from $e(v)$. When a time slot k with remaining capacity is found, v is inserted into that time slot and $s(v)$ is set to k). Otherwise, v is shifted to next time slot $k + 1$, relaxing the window constraint if necessary. The routine is repeatedly applied to all vehicles as described in the Alg. 3.

Analyzer: Calculate Blame For `calculate_blame`, the time slots assigned by the constructor are then analyzed to assign a *blame* factor to all vehicles of current solution.For our purposes, *blame* is defined as

$$blame(v) = \max[0, s(v) - l(v)]. \quad (4)$$

If $blame(v)$ is larger than zero, it indicates that v violates its window constraint. In such case, the $s(v)$ is a flaw in current solution, and it should be scheduled first in next construction. In contrast, if $blame(v) = 0$, then $s(v)$ is a valid assignment obeying both constraints. Vehicles with small $blame(v)$ are scheduled later to leave more spaces to the conflicted vehicles. $score$, the summation of all positive $blame(v)$, is calculated to determine if SWO finds a feasible solution.

Algorithm 3 `greedy_insert`(P, c): Input a prioritized sequence and upper bound to form a solution.

$S \leftarrow \emptyset$
$C \leftarrow$ an array of size T_{rush} initialized to c
for $v \in P$ **do**
  $start \leftarrow e(v)$
  while $start < T_{rush}$ and $C(start) \leq 0$ **do**
    $start \leftarrow start + 1$
  end while
  $s(v) \leftarrow start$
  $C(start) \leftarrow C(start) - 1$
end for

Prioritizer: Re-Prioritize Sequence The analyzer identifies the set of vehicles whose assignments are flawed (i.e., the set of vehicles v with positive $blame(v)$. `prioritize_seq` takes this set as input and reorders the sequence in which vehicles are scheduled by the constructor to give this subset first priority. Specifically, all vehicles with positive $blame(v)$ are first removed from the current vehicle sequence. This subset of vehicles is then sorted in descending order of $blame(v)$ and the resulting subsequence is placed at the front of the remaining current vehicle sequence to produce the re-prioritized vehicle sequence.

Shrinking the Capacity Constraints The core SWO steps above are repeated until a feasible solution is found (i.e., $score = 0$). After finding a solution without violating any constraints, the upper bound c of the capacity constraints is further reduced by δc. Then, `iterative_swo` reapplies the core SWO procedure again. A feasible solution cannot be found after Max trials, it will return the last feasible solution found and the larger upper bound c.

Experiments

Simulation Platform

The microscopic traffic simulation platform SUMO (Simulation of Urban MObility) (Krajzewicz et al. 2012) is used to evaluate the performance impact of our schedule-driven approach. To provide a realistic setting, we import the Pittsburgh map data from Open Streets into SUMO as shown in Fig. 3 and assign the Fort Pitt tunnel as the bottleneck. The first step of simulation is to generate vehicles randomly within several main residential zones of Pittsburgh. Furthermore, those vehicles must pass through the bottleneck to create the scenario of traffic congestion. Second, each driver has a flexible time window with random size, which defines the window constraints. Finally, input the departure times, routes and temporal flexibility into the proposed algorithms and generate the scheduled departure times. The flow chart of simulation is described in Fig. 4.

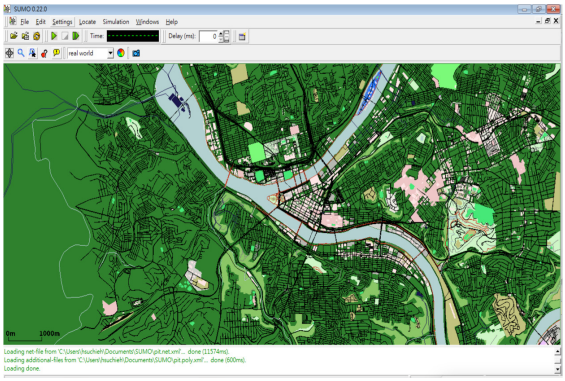

Figure 3: SUMO with realistic Pittsburgh map data

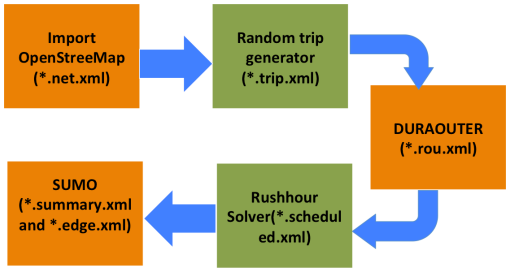

Figure 4: Flow chart of simulation

Pittsburgh Traffic Problem

To analyze the proposed algorithms, experiments are conducted using the iterative SWO, iterative min-conflicts and standard min-conflicts procedure. By standard min-conflicts we simply mean a variant that simply solves for a fixed capacity bound c. In the experiments, the value c used for fixed upper bound of standard min-conflicts and initial value of the iterative algorithms is calculated by using the timing of traffic signal located at the inbound exit of the Fort Pitt Tunnel: $c = \frac{Vm}{L(m+n)}\Delta T$, where V is the maximum speed allowed for each vehicle, and L is vehicle's length. m and n are the duration of green light and red light in one traffic signal cycle. The term $\frac{Vm}{L(m+n)}\Delta T$ represents the maximum number of vehicles passing through the bottleneck with maximum speed.

Other than the number of vehicles, the reduction of travel time depends on the expected value of window size as well. To determine the expected window size, the duration of rush hours should be defined first. A simple way to define it is using standard deviation σ of the arrival time at the bottleneck. We can define the $2a\sigma$, where a is a number greater than 1, as the duration of rush hours within which the percentage of vehicles can be estimated, e.g., 2σ and $a = 1$ correspond to 68.27% and 4σ and $a = 2$ correspond to 95.45%. After applying demand shift, we expect the variance of the arrival times becomes larger by at least $2E[l(v) - v(v)]$. Therefore, the number of scheduled vehicles during rush hours is represented as

$$N = c \times (2a\sigma + 2E[l(v) - v(v)]), \tag{5}$$

and the expected window size is calculated through

$$E[l(v) - v(v)] = \frac{N - c \times 2a\sigma}{2c} \tag{6}$$

when we have the information including the number of scheduled vehicles N, standard deviation of arrival times, and the upper bound c of capacity constraints.

These methods are compared with a baseline case where each vehicle starts at its $e(v)$. For each scheduled vehicle, the chosen window size range is between 5 minutes and 15 minutes. We ran the test with $\Delta T = 30$ second and an initial upper bound $c = 15$ vehicles per unit time ΔT. Table 1 shows the performance of three algorithms running over 400 randomly generated vehicles passing through the tunnel. The evaluation criteria are the average travel time for each vehicle (Time), the improvement of the average travel time compared to the baseline case (improvement), the final upper bound of the capacity constraints (Upper bound), the delay from the earliest departure time $e(v)$, and the overall computation time (CPU).

From the table, we can observe that the iterative SWO generates the most diverse distribution and the lowest upper bound. It is seen to shorten the commute time by 24%, which is almost 10% better than the iterative min-conflicts algorithm (at smaller computational cost). The average delay in driver departure time required to achieve this percentage improvement was just 12.8 minutes. If larger window size were provided, the distribution of arrival times would be further dispersed and provide higher gain. Another observation is that iterative SWO converges rapidly due to the constant running time of the greedy algorithm. Moreover, use of the window size heuristic to initialize priority also contributes to find a feasible solution quickly.

Next the performance under different number of vehicles is given and evaluated by 1)the mean travel time of vehicles 2) optimal upper bound of each algorithm. The performance for each algorithm are plotted in Fig. 5 and Fig. 6. We can observe that the random assignment (using a random departure time within each time window) has a comparable performance with standard min-conflicts.

Table 1: Comparison of different algorithms for 400 vehicles case

	Min-conflicts	Iterative min-conflicts	Iterative SWO
Time(s)	2931	2808	2487
Travel time improvement (%)	10.69	15.12	24.22
Upper bound	15	13	9
Avg. delay(min)	7.13	6.38	6.43
CPU (s)	399.06	213.49	14.05

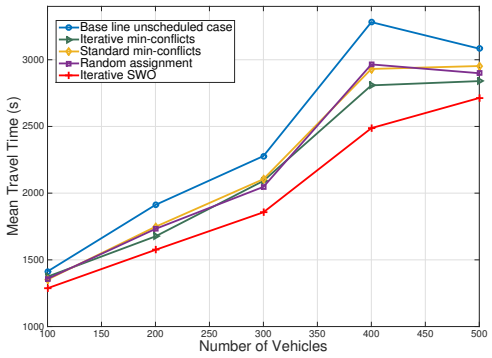

Figure 5: Mean travel time under different number of vehicles

As the number of vehicles increases, It is interesting to observe that the improvement of the 400 vehicles case is greater than the 500 vehicles case. This is because the 500 vehicles case has tighter capacity constraints, i.e., less temporal flexibility at a higher congestion level. In such case, the proposed methods still find a feasible assignment that improves traffic conditions, but the traffic congestion still exists.

Large Scale Problem

In this experiment, we extend the scale of the simulation to the range of $1000 - 2000$ vehicles per half hour, which is more realistic. According to (6), the expected value of window size is increased for getting better performance. It should be noted that only iterative SWO is able to support such large scalability, and therefore we only consider its performance in large scale setting. Due to the large window size, the improvement could be up to 27% as shown in Table 2. It is substantial in terms of traffic congestion reduction. On the other hand, the travel time improvement become marginal as the number of vehicles increases, but they are still greater than 16%. Fig. 7 displays variation of mean travel time with different number of vehicles. For large scale setting, iterative SWO shows its stability of performance again.

Penetration Rate Problem

Understanding the performance under different penetration rates is an essential issue for deploying the proposed methods under realistic scenarios. In the results presented thus

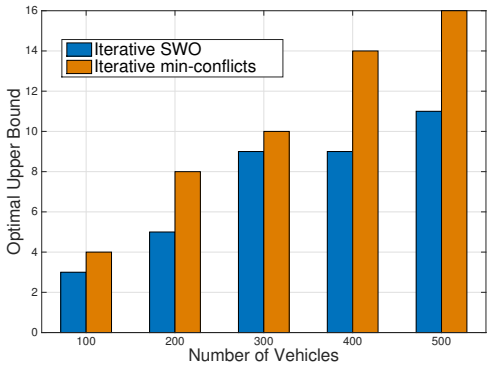

Figure 6: Minimized upper bound of the capacity constraints under different number of vehicles

Table 2: Comparison of iterative SWO algorithm for large scale setting

Num. of cars	1000	1250	1500	1750	2000
Travel time improvement (%)	27.37	25.2	22.03	19.59	16.65
Upper bound	7	9	10	11	12
Avg. delay(min)	24	21.5	22	22	22.5
Expected window (min)	27	28	28	29	29
CPU (s)	18.16	20.55	24.09	25.68	30.64

far, we have assumed that all vehicles are scheduled by the proposed algorithms. However, it may be difficult in practice to achieve such a high penetration rate. Fig. 8 presents the mean travel time of iterative SWO under different penetration rates. We assume that the window information of those scheduled vehicles are known. In order to help the scheduled vehicles to avoid rush hours, the approximation of rush hours is also assumed to have as well. It can be estimated by monitoring the tunnel daily. Then the vehicles are scheduled to avoid the estimated rush hours with other unscheduled vehicles. For example, penetration rate 40% indicates that 40% of vehicles are scheduled according to their demand and the other 60% start at their $e(v)$. The results show that the proposed algorithm can still provide a considerable 18% improvement compared with the baseline case when the penetration rate is 50%. The improvement drops to 15% at a penetration rate of 35% which still represents a reasonable reduction in overall congestion. If drivers provide larger windows for their departure times, we believe performance at lower penetration rates will improve further.

Conclusions

In this work, we describe a demand shifting traffic scheduling problem for relieving traffic congestion. Two iterative algorithms are developed to solve it under the realistic map data of Pittsburgh. Since the level of congestion directly relates to the upper bound of the capacity constraints, the proposed iterative algorithms minimize the bound to spread vehicles' departure time properly. We demonstrate that the al-

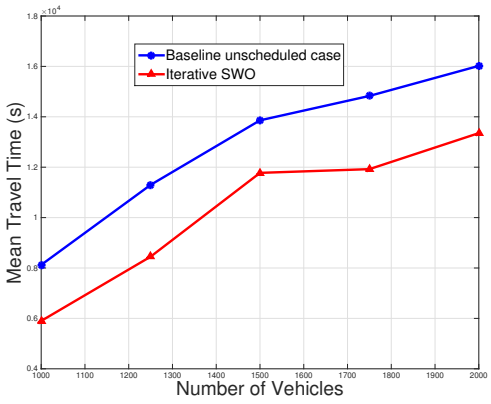

Figure 7: Performance comparison between the iterative SWO and the baseline case in large scale setting

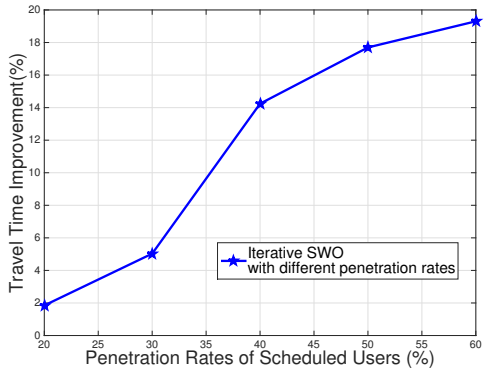

Figure 8: Performance comparison between the iterative SWO and the baseline case under different penetration rates

gorithms improve travel time efficiently compared with the baseline case. Furthermore, our solutions provide substantial gain under small penetration rates. Future work will focus on the design of coordination mechanisms for multiple interdependent bottleneck road segments.

References

Boillot, F.; Midenet, S.; and Pierrelée, J.-C. 2006. The real-time urban traffic control system cronos: Algorithm and experiments. *Transportation Research Part C: Emerging Technologies* 14(1):18–38.

Burkard, R. E.; Dlaska, K.; and Klinz, B. 1993. The quickest flow problem. *Zeitschrift für Operations Research* 37(1):31–58.

Cheng, C.-C., and Smith, S. F. 1997. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research* 70:327–357.

Downs, A. 1962. The law of peak-hour expressway congestion. *Traffic Quarterly* 16(3).

Duranton, G., and Turner, M. A. 2011. The fundamental law of road congestion: Evidence from us cities. *The American Economic Review* 2616–2652.

Even, C.; Pillac, V.; and Van Hentenryck, P. 2015. Convergent plans for large-scale evacuations. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15)*.

Ford Jr, L. R., and Fulkerson, D. R. 1958. Constructing maximal dynamic flows from static flows. *Operations research* 6(3):419–433.

Han, D.; Yang, H.; and Wang, X. 2010. Efficiency of the plate-number-based traffic rationing in general networks. *Transportation Research Part E: Logistics and Transportation Review* 46(6):1095–1110.

Helly, W. 1900. Simulation of bottlenecks in single-lane traffic flow.

Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 353–373.

Kim, S.; Shekhar, S.; and Min, M. 2008. Contraflow transportation network reconfiguration for evacuation route planning. *Knowledge and Data Engineering, IEEE Transactions on* 20(8):1115–1129.

Köhler, E.; Langkau, K.; and Skutella, M. 2002. Time-expanded graphs for flow-dependent transit times. In *AlgorithmsESA 2002*. Springer. 599–611.

Krajzewicz, D.; Erdmann, J.; Behrisch, M.; and Bieker, L. 2012. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements* 5(3&4):128–138.

Li, J.-Q.; Mirchandani, P. B.; and Borenstein, D. 2009. Real-time vehicle rerouting problems with time windows. *European Journal of Operational Research* 194(3):711–727.

Liu, W.; Yang, H.; and Yin, Y. 2014. Traffic rationing and pricing in a linear monocentric city. *Journal of Advanced Transportation* 48(6):655–672.

Meyer, M. D. 1999. Demand management as an element of transportation policy: using carrots and sticks to influence travel behavior. *Transportation Research Part A: Policy and Practice* 33(7):575–599.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58(1):161–205.

Pluntke, C., and Prabhakar, B. 2013. Insinc: A platform for managing peak demand in public transit. *JOURNEYS, Land Transport Authority Academy of Singapore*.

Ramasesh, R. 1990. Dynamic job shop scheduling: a survey of simulation research. *Omega* 18(1):43–57.

Thiagarajan, A.; Ravindranath, L.; LaCurts, K.; Madden, S.; Balakrishnan, H.; Toledo, S.; and Eriksson, J. 2009. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 85–98. ACM.

Viegas, J. M. 2001. Making urban road pricing acceptable and effective: searching for quality and equity in urban mobility. *Transport Policy* 8(4):289–294.