Unspread the Jam: Scheduling Traffic Lights to Reduce Congestion

Yoav Levi, Ayal Taitler, Isaac Keslassy
Technion

***Abstract*—In this paper, we consider the practical problem of scheduling traffic lights to reduce the average vehicle waiting times. We find that existing scheduling algorithms have lackluster performance. Instead, we introduce two algorithms. First, *extended CMSM (eCMSM)*, which extends CMSM from a switch scheduling model to a general traffic-light scheduling model. We prove that eCMSM can optimally schedule any traffic batch. Second, we introduce *Front-Pressure (FP)*, which aims to further reduce the average waiting time at general intersections. We then evaluate empirically these two algorithms. We find that when using them, the best average waiting time can be improved in 98% of the simulations when compared to several existing algorithms, most significantly in congested settings.**

I. Introduction

Motivation. Traffic congestion in signalized intersections is a major challenge, especially in dense urban areas, where congestion at one intersection may spread to neighboring intersections. The long waiting times of the vehicles directly affect the productivity of the passengers, and result in annual cost penalties of at least hundreds of billions of dollars [1].

Nowadays, most traffic lights are controlled offline using a predetermined policy. This policy sets a specific time sequence of traffic-light *phases* (*i.e.,* allowed configurations). However, improved sensors in signalized intersections can now estimate in real-time the state of the intersection and enable online policies [2], with a major potential impact on the economy [3]. Ideally, an online state-aware policy would optimize the average waiting time across a whole city, but this proves impractical. Therefore, we focus on a single intersection.

Related work. Scheduling intersections has been the focus of much recent research in the field of traffic control [4]. A notable algorithm is the *Max-Pressure* algorithm [5], which maximizes the number of vehicles in lanes given a green light. The phases can be either hand-crafted or computed automatically [6]. Other approaches such as Stochastic Planning [4], Linear Programming [7] and Machine Learning [8] have also been applied to the problem of traffic control.

In addition, the problem of finding a scheduling policy in traffic intersections is closely related to that of packet scheduling in computer networks. Network routers also need to schedule packets that are destined to different router ports, such that there is no collision between packets. However, scheduling in road intersections involves additional constraints; *e.g.,* in each scheduled phase, lanes should not cross each other to avoid vehicle collision. In particular, the general *Back-Pressure* (BP) algorithm [9], also known as Maximum Weight Matching (MWM) in a limited switch setting, is a similar algorithm to Max-Pressure. It was shown to achieve 100% throughput under independent arrival processes with admissible arrival rates. Unfortunately, it can also cause significant starvation and unfairness among queues, especially with reactive traffic [10]–[13]. Another common scheduling approach is Maximum Size Matching (MSM), which maximizes the instantaneous throughput of the switch by choosing a set with the highest number of occupied queues at each time-slot. Although it maximizes the instantaneous throughput, MSM does not achieve 100% throughput in general and may be unstable for specific rates of arrival [14]. Another related problem is Time Slots Assignment (TSA) in satellite switching. TSA periodically accumulates traffic and then schedules each resulting batch without considering ongoing arrivals. Critical Maximum Size Matching (CMSM) was proved to solve the TSA problem by reaching a schedule that needs a minimal number of time-slots [15]. Furthermore, like BP, CMSM achieves 100% throughput for any admissible arrival rates under batch scheduling [16].

Contributions. In this paper, we look for *a practical online scheduling algorithm that reaches a low average waiting time*.

In Sec. II, we start by defining our traffic intersection model by relying on its *conflict graph*, which represents the sets of lanes that cannot be scheduled simultaneously.

Then, in Sec. III, we explain why MSM, BP and CMSM do not fit our goal: MSM and BP have lackluster practical performance, while CMSM is ill-defined for the traffic intersection model. Instead, we can exploit the small size of the single-intersection problem by exploring more complex algorithms. We first introduce *extended CMSM (eCMSM)*, an extension of CMSM that can handle scheduling at traffic intersections. eCMSM picks a maximum-size phase, but also makes sure to select at least one lane from each maximum-weight clique. We prove that eCMSM is optimal for the intersection equivalent of the TSA problem. Moreover, since we are looking for an even better practical performance, we introduce our *Front-Pressure (FP)* algorithm, which displays better performance under congestion. FP picks again a maximum-size phase, but among all such schedules, picks the one with the highest lexicographic weight, *i.e.,* breaks ties by giving preferential service to the phase that services the longest queue of vehicles, and in case of tie, the second-longest queue, and so on.

Finally, in Sec. IV, we evaluate and compare all these policies. Using a realistic setting, we find that FP and eCMSM achieve the best average waiting time among all algorithms in 76% and 22% of the cases, respectively.

978-1-5386-6378-3/18/$31.00

II. Model and Notations

We consider the problem of finding a scheduling policy for a general-shape signalized traffic intersection with $n \geq 2$ entrance lanes that are exclusively controlled by traffic lights.

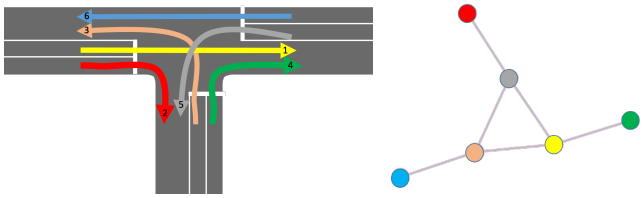
Conflict graph. As Fig. 1 illustrates, we model the intersection using its *conflict graph*, *i.e.,* an undirected graph $G(V, E)$ where each of the $|V| = n$ vertices represents a lane, and an edge between two vertices exists iff the lanes are conflicting. For instance, given the T-shaped intersection in Fig. 1a, the resulting conflict graph is displayed in Fig. 1b. Note that we can also add a pedestrian cross-walk with a pedestrian traffic light, simply by augmenting the conflict graph.

Phase. We classically assume a discrete-time model of an isolated intersection, with periodic scheduling cycles of pre-determined fixed duration T [17]. Namely, let a *phase* denote a group of lanes that can be assigned a green light at the same time without conflicts. Also define a *time-slot* as the minimal time that can be assigned to each phase; for simplicity, we normalize time so that each time-slot lasts 1. Then every cycle of T time-slots, we want to schedule a set of T traffic-light phases that will last for the entire cycle. For instance, if $T = 10$, then we can schedule one phase for 7 time-slots, then another for 3 time-slots. Denote the set of vehicles in a given lane as a *flow*. We assume that the schedule is not delayed, *i.e.,* flows start moving immediately once assigned a green light, and stop right after the red light.

Queueing. At each lane i, we denote by $W(i) \in \mathbb{N}$ the number of vehicles waiting for the green light, also referred to as *queue size* or *weight*. This number is always known, *e.g.,* via a dedicated sensor. For simplicity, we treat arriving cars as if they all arrive at the beginning of a cycle. *i.e.,* car arrivals are delayed in the model to the next beginning of a cycle. In addition, when a flow gets a green light, we assume a constant evacuation rate of 1 [car/time-slot]. We will say that a scheduling algorithm is *stable*, or *achieves 100% throughput*, whenever it stabilizes all the queues of the intersection under all independent and admissible arrival rates [16], [18]. However, in this paper, we do not restrict ourselves to admissible traffic, and are also interested in scheduling algorithms that can also perform well under non-admissible arrivals due to congestion.

MIS. Using graph theory, a phase can be seen as an *Independent Set (IS)* of the conflict graph, *i.e.,* a subset of vertices without any two adjacent vertices, thus guaranteeing that there are no conflicting flows. In addition, it is natural to schedule flows from a *Maximal Independent Set (MIS)*, *i.e.,* an IS that is not a subset of any other IS, which will achieve a greater throughput than scheduling flows from a non-maximal IS.

Problem statement. *Our main objective is to find a scheduling algorithm that reduces the average waiting time per vehicle.* By Little's Law (whenever applicable), this is equivalent to reducing the average queue occupancy.



(a) T-shaped intersection (b) Conflict graph

Fig. 1: **(a)** *T-shaped intersection*, with its **(b)** *conflict graph*. For instance, lanes 1 in yellow and 4 in green cannot be scheduled together in the same phase, because they share the same destination.

III. Algorithms

Settings. We exploit the small size of the intersection scheduling problem (typically $n \leq 20$) to introduce more complex algorithms. In particular, we assume that all the MISes are first computed offline once, *e.g.,* by applying the Bron-Kerbosch algorithm [19] on the conflict graph of the intersection. This stage is potentially the most time-consuming due to the complexity of computing the MISes, but since it is done offline, this is not considered as a major problem. Then, online, we pick a phase among all possible MISes at each cycle.

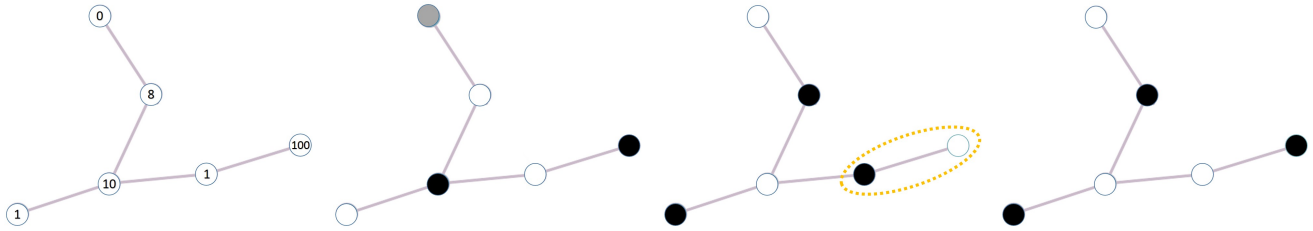
Challenges. We start by considering three well-known algorithms, which we presented in the Introduction, and explain why they do not fit our goals.

MSM. First, *Maximum Size Matching (MSM)* maximizes the instantaneous throughput by choosing a random maximum-cardinality MIS at each time-slot. Unfortunately, since it does not take the queue lengths into account, it does not prioritize a queue with 100 vehicles over a queue with a single vehicle, and therefore can result in poor performance [14].

BP. Next, *BackPressure (BP)*, also known as *Maximum-Weight Matching (MWM)*, maximizes the number of vehicles in lanes with a green light. While it provides 100% throughput, it tends to strongly favor the heaviest lanes, and does not always pick a maximum-cardinality schedule like MSM. As a result, it tends to *spread the traffic jam* across more lanes.

Fig. 2 illustrates a BP schedule example. Specifically, Fig. 2a provides an example of conflict graph and the associated queue sizes. For instance, the rightmost lane has 100 vehicles waiting for a green light. Then, Fig. 2b shows the resulting BP schedule, which strongly favors the long queues and schedules the lanes with 100 and 10 vehicles. It is not a maximum-cardinality schedule since it only schedules 2 non-empty queues instead of 3.

CMSM. Instead, we would like a hybrid algorithm that can combine the maximum-cardinality of MSM and the priority given to longer queues by BP, thus *unspreading the traffic jam* unlike BP. We start by considering the *Critical Maximum Size Matching (CMSM)* algorithm, which (in bipartite graphs) selects an MSM schedule that picks at least one element from each maximum-length (critical) source or destination. Unfor-



(a) Conflict graph and queue sizes (b) BP schedule (c) eCMSM schedule (d) FP schedule

Fig. 2: A *conflict graph* and the potential schedules produced by each algorithm in the first time-slot. Specifically: **(a)** *Conflict graph*, together with the corresponding queue sizes at each lane. **(b)** *BP phase schedule*, which maximizes the number of vehicles in lanes with a green light (here $100 + 10 + 0 = 110$), but also picks an empty lane (in grey). **(c)** *eCMSM schedule*, which makes sure to schedule at least one lane in each maximum-weight clique, and in particular picks one of the two lanes in the maximum clique in a dashed ellipse. Note that the eCMSM schedule is also always a potential *MSM schedule*. **(d)** *FP schedule*, which picks the MSM schedule with the highest lexicographic order. Here, FP makes sure to schedule the lane with 100 vehicles. Then, if it were to schedule the lane with 10, as for BP, the resulting schedule would not be maximal-size. Therefore, it schedules the lane with 8, and the remaining leftmost lane with 1.

tunately, CMSM only applies to bipartite graphs, and therefore does not work on general conflict graphs of intersections.

eCMSM. Consequently, we start by introducing *extended CMSM (eCMSM)*, which extends CMSM to intersections (and is equivalent to CMSM when reduced to bipartite graphs). We exploit the notion of a *clique*, *i.e.,* a subset of the set of vertices such that any two subset vertices are adjacent. Let a *maximum-weight clique* denote a clique with a maximum weight, *i.e.,* a clique that contains a maximum number of vehicles. Then eCMSM schedules a maximum-cardinality phase (as in MSM) such that all maximum-weight cliques have at least one scheduled representative.

For instance, Fig. 2c illustrates a potential eCMSM schedule. This schedule is maximum-cardinality, since it schedules three non-empty queues, and it also schedules a representative from the maximum-weight clique (in this case, the representative with 1 vehicle).

We can prove the following optimality result for eCMSM in the intersection equivalent of the TSA problem:

Theorem 1. *Any intersection with maximum-weight clique of weight W and no arrivals can be optimally emptied by eCMSM in W time-slots.*

Proof: We start by proving that if the weight of any maximum-weight clique is W in time-slot t and there are no arrivals, then under the eCMSM policy, the weight of any maximum-weight clique C at time-slot $t+1$ is at most $W-1$.
Case 1: C is a maximum-weight clique at time t. eCMSM services at least one representative lane from each maximum-weight clique at time t, and therefore the weight of any maximum-weight clique at $t+1$ will be reduced to $W-1$.
Case 2: C is a non-maximum-weight clique at t. Cliques that are not maximum-weight at t have a weight of at most $W-1$, and this weight cannot increase.
Finally, by induction, after W time-slots, the weight of any maximum-weight clique is 0; and therefore all queues are empty.
We conclude the proof by showing *optimality*, *i.e.,* no scheduling algorithm can outperform eCMSM: by definition, a maximum-weight clique of weight W cannot be serviced in less than W time-slots. ■

FP. Unfortunately, the eCMSM performance can sometimes still be lacking, especially in the presence of heavy congestion. For instance, in Fig. 2c, we can see how eCMSM does not service the heavily-congested queue with 100 vehicles. Therefore, we introduce our *Front-Pressure (FP)* algorithm. As detailed in Algorithm 1, at each time-slot, FP picks a maximum-cardinality MIS by breaking ties using the queue-length lexicographic order. Namely, like MSM and eCMSM, FP picks a maximum-cardinality MIS. However, if the maximum-cardinality MIS is not unique, then it picks the maximum-cardinality MIS with the longest queue. Else, if there are several such schedules, it keeps comparing the second-longest queue, third-longest queue, and so on, until reaching a unique MIS. If it has compared all queues and there are still several MISes with equal queue lengths, it picks a random one.

The complexity of the algorithm can be divided into two parts. The first part is offline and reasonable, so it is not a major problem for our purposes. The second part is online, and is governed by the complexity of the sorting problem among all potential MISes, which can be done efficiently. For instance, in the example of Fig. 1, the 6 lanes lead to 4 MISes; and with a cross-shaped intersection with 12 lanes, we get 17 MISes. With these 17 MISes, an unoptimized FP runs in some 10 milliseconds on a simple laptop.

IV. Simulations

In this section, we present empirical results for our eCMSM and FP algorithms, and compare them to other algorithms.

Settings. We simulated a general cross-shaped intersection, with three entering lanes at each direction of the cross, *i.e.,* a total of twelve entering flows.

Algorithm 1: Front-Pressure

input : conflict graph G, weight vector W, cycle time T
output: set M of all MISes, scheduling vector S

initialize S with zeros
$M \leftarrow$ Find all MISes of G ▷ computed offline
while *intersection is not empty* & $\sum_i S_i < T$ **do**
  $W \leftarrow$ Get remaining weights given S
  $\text{size}(m) \leftarrow m \cdot \mathbb{1}_{W>0}, \ \forall m \in M$ ▷ *i.e.,* inst. throughput
  $M' \leftarrow$ sort M by size
  $P \leftarrow \{m' \in M' \mid \text{size}(m') = \max_{m \in M'} \text{size}(m)\}$ ▷ MSM
  if $|P| > 1$ **then**
    $P' \leftarrow$ Find (lexicographic) set of maxima in $\{\text{sort}(m \odot W) | m \in P\}$ ▷ sorted element-wise product
  if $|P'| > 1$ **then**
    $p \leftarrow \text{random}(P')$ ▷ pick random max element
  $i \leftarrow$ index of p in M
  $S_i = S_i + 1$.
return M, S

Algorithms. We compared the results to three other algorithms: *Round-Robin* [6], *MSM* [14], and *BP* [9]. RR is a static scheduling policy, similar to what is commonly used today, that repeats periodically a fixed number of phases with predetermined green times. This policy is oblivious of the current queue sizes at the traffic intersection. In particular, in our implementation, RR cycles in round-robin through the MISes and allocates the same time for each MIS.

Arrivals. For each simulation, we consider a vector of arrival rates at each lane. At the beginning of each cycle, the actual number of vehicles for each lane is sampled from a Poisson distribution with mean corresponding to the lane's arrival rate.

Measures. All the congestion plots present the number of cars queued at the intersection as a function of the cycle number. For each of the first two plots, we averaged ten simulations. The bold line presents the mean for each algorithm, and the standard deviation is given as the transparent envelope of the line.

Admissible traffic. The first plot (Fig. 3) illustrates a stable setting with admissible rates, *i.e.,* the intersection can serve all the vehicles without causing congestion. The cycle time for this simulation was 120 time-slots, and the arrival rates per cycle were $[(33, 9, 5), (5, 24, 28), (38, 14, 33), (47, 9, 14)]$, where each of the four triplets represents the rates from each of the four source roads to each of the three other destination roads, in the same order. Both RR and MSM are unstable and experience a near-linear increase of the queue lengths over time, while BP, eCMSM and FP keep a stable behavior, with eCMSM having a slightly higher number of vehicles.

Non-admissible traffic. The second plot (Fig. 4) illustrates an unstable setting with non-admissible rates, *i.e.,* vehicles are arriving too quickly for any intersection schedule to evacuate them all, so congestion is building up at the intersection. The cycle time for this simulation was 120 time-slots, and the arrival rates per cycle were $[(39, 11, 6), (6, 28, 34), (45, 17, 39), (56, 11, 17)]$. As expected,

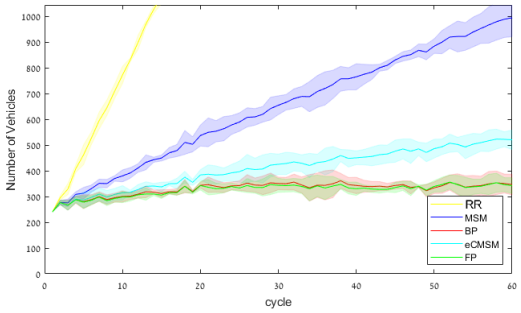


Fig. 3: Simulation with admissible rates.

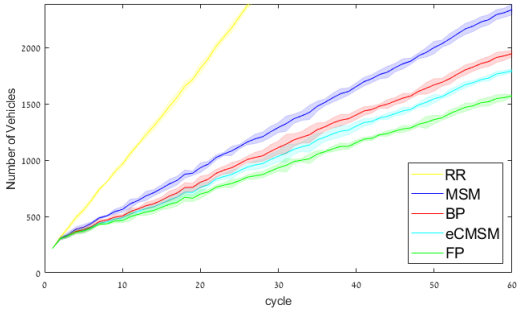


Fig. 4: Simulation with non-admissible rates.

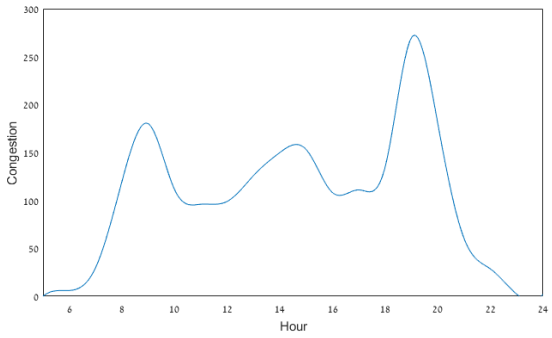
congestion builds up in all algorithms. FP builds up the least congestion, maintaining the shorter average queue size and therefore shorter waiting time (by Little's Law); and eCMSM displays the second-best performance.

Throughout the day. The last evaluation attempts to illustrate a real-world scenario. In a real intersection, congestion usually occurs in the hours of the day where people go to work or get back from work. As shown in Fig. 5a, we start from a real-world daily congestion graph of Mexico City obtained from [20]. We then model a cross-shaped intersection with twelve flows, where the arrival rates at each cycle are normalized by making them proportional to the congestion value in the corresponding hour. Namely, there is more congestion at 9:00 and 19:00, and less at 6:00.

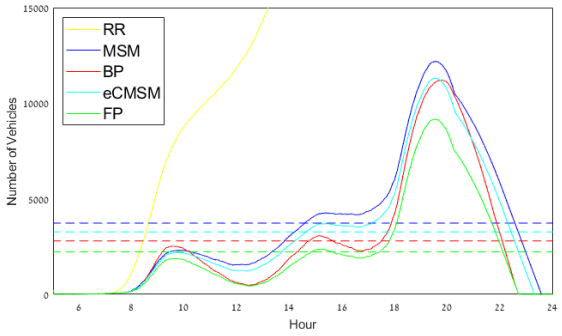
We tested this realistic pattern of congestion over 2,000 simulations. In each simulation the vector of arrival rates was sampled from an *i.i.d.* uniform distribution, and the average waiting time was calculated for each algorithm. Fig. 5b plots the results of one simulation that showed more significant differences, and Table I synthesizes the results of all simulations.

Fig. 5b shows how the general behavior of all algorithms is the same over the day (excluding RR, which behaves particularly badly). FP clearly exhibits the best performance, as can be seen most clearly in the peak evening hours. The dashed straight lines represent the average number of vehicles over the whole simulation. FP achieves the smaller average number of vehicles and the shorter average waiting time.

Table I provides the average waiting times over the 2,000 simulations. As expected, RR performs the worst. Then,



(a) Measured vehicle congestion in Mexico City.



(b) A Mexico-City simulated intersection under different policies.

Fig. 5: Simulation based on Mexico City real-world data.

Policy	Average waiting time
RR	109.08
MSM	15.55
BP	12.62
eCMSM	12.42
FP	**12.11**

TABLE I: Average statistics over 2,000 simulations

MSM, BP, eCMSM and FP provide increasingly better average performance. eCMSM manages to improve the waiting time over BP by 1.6%, and FP outperforms BP by 4.2%. This relatively-small apparent increase is an average between many simulations with little congestion and negligible improvement, and a few simulations with a larger congestion and a more distinct improvement (as in Fig. 5b). In fact, FP is the best policy in 76% of all simulations, eCMSM in 22% of all simulations, and BP in the remaining 2%.

V. Conclusion

In this paper, we used a graph-based model for traffic intersections, and computed the phases automatically based on the intersection model. We presented two scheduling algorithms for traffic lights at an isolated intersection, and compared their performance to other known algorithms. The first algorithm, eCMSM, is an extension of CMSM to intersection settings, and extends the notion of a critical source or destination in a bipartite graph to a critical clique in a general graph. We further prove that eCMSM can optimally schedule any traffic batch. Our second algorithm, FP, maximizes the instantaneous throughput, but also has a tie-breaking mechanism that favors longer queues. We showed how eCMSM and especially FP outperform existing algorithms in our evaluations, especially in heavily congested intersections. We also argued why we believe that they are ready for implementation in real intersections.

Acknowledgments

The authors would like to thank Shay Vargaftik, Pedro Mercader, Jack Haddad and Erez Karpas for their assistance and invaluable discussions during this project.

This work was partly supported by the Hasso Plattner Institute Research School, the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel Cyber Bureau.

References

[1] G. Cookson and B. Pishue, "Global traffic scorecard," Tech. Rep., 2017.
[2] H. Li, N. Chen, L. Qin, L. Jia, and J. Rong, "Queue length estimation at signalized intersections based on magnetic sensors by different layout strategies," *Transportation res. proc.*, vol. 25, pp. 1626–1644, 2017.
[3] M. Cassini and R. Wellings, "Seeing red: Traffic controls and the economy," *IEA*, 2016.
[4] B. Yin, M. Dridi, and A. El Moudni, "Markov decision process for traffic control at an isolated intersection," in *IEEE ICTAI*, 2013, pp. 789–794.
[5] P. Varaiya, "Max pressure control of a network of signalized intersections," *Transportation Research Part C*, vol. 36, pp. 177–195, 2013.
[6] A. Ghavami, K. Kar, and S. Ukkusuri, "Delay analysis of signal control policies for an isolated intersection," in *IEEE ITSC*, 2012, pp. 397–402.
[7] P. Mercader, M. Ornik, P.-O. Gutman, and I. Ioslovich, "Optimal signal timing for multi-phase intersections," *Report*, 2018.
[8] S. Araghi, A. Khosravi, M. Johnstone, and D. Creighton, "Q-learning method for controlling traffic signal phase time in a single intersection," in *IEEE ITSC*, 2013, pp. 1261–1265.
[9] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.
[10] A. Shpiner and I. Keslassy, "Modeling the interactions of congestion control and switch scheduling," *Computer Networks*, vol. 55, no. 6, pp. 1257–1275, 2011.
[11] P. Giaccone, E. Leonardi, and F. Neri, "On the interaction between TCP-like sources and throughput-efficient scheduling policies," *Performance Evaluation*, vol. 70, no. 4, pp. 251–270, 2013.
[12] H. Seferoglu and E. Modiano, "TCP-aware backpressure routing and scheduling," *IEEE Trans. on Mobile Computing*, vol. 15, no. 7, pp. 1783–1796, 2016.
[13] S. Vargaftik, I. Keslassy, and A. Orda, "No packet left behind: Avoiding starvation in dynamic topologies," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2571–2584, 2017.
[14] I. Keslassy, R. Zhang-Shen, and N. McKeown, "Maximum size matching is unstable for any packet switch," *IEEE Communications Letters*, vol. 7, no. 10, pp. 496–498, 2003.
[15] T. Weller and B. Hajek, "Scheduling nonuniform traffic in a packet-switching system with small propagation delay," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 813–823, 1997.
[16] S. Iyer and N. McKeown, "Maximum size matchings and input queued switches," in *Allerton*, vol. 40, no. 2, 2002, pp. 615–625.
[17] F. Mannering, W. Kilareski, and S. Washburn, *Principles of Highway and Traffic Engineering*. John Wiley & Sons, New York, 2008.
[18] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," *IEEE Infocom*, vol. 2, pp. 792–799, 1998.
[19] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Comm. of the ACM*, vol. 16, no. 9, 1973.
[20] "Tomtom traffic index," https://www.tomtom.com/en_gb/trafficindex/.