Swarm Optimization Methods for Traffic Light Cycle Control
Applied Mathematics 221 Final Project

Robert Chen
robertchen@college.harvard.edu

Alex Wang
alexwang@college.harvard.edu

August 31, 2017

Abstract: Well-timed traffic light can tremendously decrease vehicles' time spent at red lights, but timing the lights well is a very complex, difficult problem to solve. In this paper, we explore swarm optimization methods for optimizing traffic light behavior, which allow us to minimize a function with knowing its analytic form. We apply particle swarm optimization (PSO) and a novel formulation of ant colony optimization (ACO). In our simulation, both these methods yield significant decreases in average total travel time for vehicles, an improvement of 7% over baselines. Our adaptation of ACO performs particularly well, suggesting its potential application in solving optimization problems in a variety of other fields.

1 Introduction

Traffic lights are a fixture of everyday life with seemingly simple governing rules but with complex effects on traffic patterns. As thousands of hours are wasted everyday waiting in slow traffic or for red lights, a question of great practical application is how to design a traffic light's behavior or a system of traffic lights so as to minimize the amount of time wasted in traffic. Ideally, we would want to model the problem formally to guide the optimization process and get theoretical guarantees on any solution found. However, in general it is very difficult to model a traffic system, as there are a huge number of variables: road layout, traffic volume, traffic light behavior, etc. Therefore, we attempt to optimize without doing so.

Particle swarm optimization and ant colony optimization are two optimization techniques that fall into the broad category of metahueristics, and more specifically into swarm optimization. As such, they make no assumptions about the function to be optimized, but come at the cost of having no guarantees on the quality of solution found or time needed. Both approaches utilize a group of agents, a "swarm", that each explore the solution space and interact with each other to search intelligently.

The rest of this paper is organized as follows: we describe the problem of traffic optimization quasi-formally and previous attempts to solve this problem. We then formally define particle swarm optimization and ant colony optimization. Finally, we describe our experiments and discuss results.

2 Related Work

The problem of traffic light optimization has been an active area research, with focuses on different granularities ranging from control of a single light[1] to a macroscopic city traffic network[2]. The former is significantly easier to model, while for the latter, which is the problem that we address, to the best of our knowledge has no standard model.

There has been significant focus on applying reinforcement learning ideas to large-scale traffic light optimization[3][4], as well as some work on applying genetic algorithms[5] and neural networks. These papers unsurprisingly report that their approaches are able to reduce mean waiting time as compared to various baselines. Notably, all of these approaches do not require that we are able to write down a model for the traffic system and lights; they only require that the function can be evaluated, suggesting that our approach is equally valid.

These swarm optimization methods have had broad applications since their conception. For example, they are used in a variety of industries such as manufacturing and electricity control[6]. Most notably, particle swarm optimization was used to determine the parameters for artificial neural networks, before backpropagation became the standard. Indeed, we have

[1] DD98.
[2] Wie+04.
[3] Wie+04.
[4] Ste+05.
[5] SGR04.
[6] ES01.

encountered some works that use particle swarm optimization to optimize a model of a real world traffic network[7][8], and to some extent our particle swarm experiments replicate their approach, but under different assumptions. However, as ant colony optimization is generally applied in a graph setting, we have not encountered any previous work that applies ant colony optimization for traffic light optimization.

3 Problem Description

As we are interested in optimization techniques for arbitrary traffic systems, we only seek to optimize the network with respect to the behavior of the traffic lights in the system. Thus, for this problem we make no assumptions about the layout of the roads, volume of traffic, etc. Formally, we model this problem as follows: we model the traffic light behavior as an n-dimensional vector $\boldsymbol{x} \in \mathbb{R}^{m \times n}$, where we have n features that determine a traffic light's behavior and m traffic lights in the system. For example, a straightforward feature representation is to have a 4 phase light cycle, and a single light is represented by 4 numbers, each corresponding to the duration of a phase in the cycle. If our traffic system has 25 lights, then a behavior vector for this entire traffic light system is in $\mathbb{R}^{25 \times 4}$. To evaluate the "goodness" of a behavior vector $\boldsymbol{x}$, we have a function $f : \mathbb{R}^{m \times n} \to \mathbb{R}_+$ which takes in a traffic light behavior and outputs the mean waiting time for a car in this system. We want to find

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

Note that we make no assumptions about the analytic form of f and make no attempts to model the underlying traffic system. Instead, we take f to be some sort of black box function that can be evaluated, but we know nothing else about it. As such, we have very little to work with in optimizing f, and therefore must turn to extremely general optimization methods.

4 Particle Swarm Optimization (PSO)

4.1 Description

Particle swarm optimization (PSO) is a metaheuristic optimization algorithm that makes very few assumptions about the function to be optimized, and thus is suitable for solving a broad class of functions, including complicated non-convex function. Inspired by the social behavior of various animals, PSO uses a swarm of particles, each associated with a position in the solution space and a velocity. The algorithm consists of alternating between calculating the velocity based on the performance found across all neighbors and updating the position based on the particle's velocity. The particles tend to cluster together in the search space[9], thereby forming a swarm. There exist many other variants of swarm optimization, each inspired by different biological behaviors, but we describe the basic algorithm here:

procedure PSO($f, N, T, \boldsymbol{b}_l, \boldsymbol{b}_u, c_0, c_1$)
  for $i = 1, N$ **do**
    $\boldsymbol{x}_i \sim U(\boldsymbol{b}_l, \boldsymbol{b}_u)$
    $\boldsymbol{v}_i \sim U(-|\boldsymbol{b}_u - \boldsymbol{b}_l|, |\boldsymbol{b}_u - \boldsymbol{b}_l|)$
    $\boldsymbol{p}_i = \boldsymbol{x}_i$
  $\boldsymbol{g} = \arg\max_n f(\boldsymbol{x}_n)$
  for $t = 1, T$ **do**
    for $i = 1, N$ **do**
      $r_0, r_1 \sim U(0, 1)$
      $\boldsymbol{v}_i = \boldsymbol{v}_i + c_0 \times r_0 \times (\boldsymbol{p}_i - \boldsymbol{x}_i) + c_1 \times r_1 \times (\boldsymbol{g} - \boldsymbol{x}_i)$
      $\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$
      if $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_i)$ **then**
        $\boldsymbol{p}_i = \boldsymbol{x}_i$
        if $f(\boldsymbol{x}_i) < f(\boldsymbol{g})$ **then**
          $\boldsymbol{g} = \boldsymbol{x}_i$
  return $\boldsymbol{g}$

Here, we want to optimize $f : \mathbb{R}^n \to \mathbb{R}$. There are N particles, each with position $\boldsymbol{x}_i \in \mathbb{R}^n$ and velocity $\boldsymbol{v}_i \in \mathbb{R}^n$. We also keep track of $\boldsymbol{p}_i$, the best found solution for particle i, and $\boldsymbol{g}$, the best found solution by any particle. We randomly initialize the positions and velocities uniformly on the search space. For a fixed number of iterations (or while $f(\boldsymbol{g})$ is not close to some value), we alternate between computing a new velocity as a randomized linear combination of the current velocity,

[7] KB09.
[8] GOA13.
[9] Ken11.

the distance from the local best $\boldsymbol{p}_i$ and the current position $\boldsymbol{x}_i$, and the distance from the global best $(\boldsymbol{g} - \boldsymbol{x}_i)$. By updating the velocity as a function of the distance from the globally best solution, PSO encourages particles to explore areas near the best solution. Parameters c_0, known as the cognitive parameter, and c_1, known as the social parameter, control to what degree particles should explore near the locally and globally best solutions respectively; typically this is set to $c_0 = c_1 = 2$[10].

4.2 PSO for Traffic Lights

Particle swarm optimization pairs nicely with traffic light optimization problem. The main decision is how to represent a traffic light's behavior as a vector of real numbers. An intuitive formulation we explore is to fix the order of the n light phases in the cycle, and then represent each traffic light as a vector $\boldsymbol{x} \in \mathbb{R}^n$ where each component $\boldsymbol{x}_i$ represents the length of the i-th phase. However, we observe that with this formulation, it doesn't make sense for a length to be of negative duration. Furthermore, in practice, traffic lights stay on a phase at most roughly a few minutes, depending on the intersection, and at least a few seconds. Thus, for our simulations, we impose a restriction that each component must be within some minimum time t_{min} and maximum time t_{max}, i.e. $\boldsymbol{x} \in [t_{min}, t_{max}]^n$. This fits naturally into the PSO algorithm as the algorithm allows us to bound the search space we want to explore. We describe our specifications for t_{min}, t_{max} in later sections.

Another possible representation of a traffic light includes the initial phase in the cycle of light phases. To do this, we extend $\boldsymbol{x}$ to be in $\mathbb{R}^{n+1}$, where the extra component tells us which of the n phases the cycle starts at. The difficulty here is that intuitively this would be represented as a discrete value, i.e. $\boldsymbol{x}_{n+1} \in [n]$, but PSO assumes continuous values. To accommodate for this, we let the initial phase be a continuous value in some range, e.g. $[t_{min}, t_{max}]$, and then round to the nearest $t_{min} + \frac{i}{n}(t_{max} - t_{min})$. This is a general method that allows us to use PSO for discrete valued inputs. It is not immediately clear what the interpretation of this discrete-to-continuous relaxation is, but one possible explanation is that the continuous value is a measure of how certain the algorithm is of assigning that discrete value.

5 Ant Colony Optimization (ACO)

5.1 Description of ACO

Ant Colony Optimization (ACO) is a heuristic method for finding optimal paths in an undirected graph. In ACO, each edge has a pheromone value associated with it, with higher pheromone values roughly correlating with edges that are more useful in constructing the optimal solution (such as shortest path or path that minimizes the sum of cost associated with edges). At each iteration t, multiple "ants" traverse the graph to each construct a solution. To construct each solution, an ant starts at some vertex i of the graph and pick some "legal" edge toward vertex j to traverse randomly, with probability:

$$P_{ij}(t) \propto (T_{ij}(t))^a * h_{ij}^b$$

Here we have $T_{ij}(t) =$ pheromone value of edge (i, j) at time t, $h_{ij} =$ some heuristic value associated with edge (i, j) that stays constant over time, and a and b fixed constants. Once a neighbor is chosen and we reach vertex j, we continue to "build" a solution by traversing additional edges, each time choosing an edge by the above probability rule. What edges are defined as "legal" is application-specific. The pheromone values are initialized to be the same for all edges at the start of the algorithm.[11]

Once a set of solutions is obtained during iteration t, we evaluate the quality/performance of these solutions and update the pheromone values. For each edge (i, j), we update $T_{ij}(t+1) = (1-p) * T_{ij}(t) + \sum_{z \in Z} f(z)$, for Z the set of solutions in iteration t that include edge (i, j) and $f(z)$ some function of the performance of solution z. The parameter $0 < p < 1$ controls the rate of evaporation of pheromone over time. Essentially, "good" solutions will contribute additional pheromone to all of its edges, and so these edges will then be more likely to be used again when constructing solutions during the next iteration.

5.2 Traffic Light ACO Algorithm

For many problems, such as the canonical example Traveling Salesman Problem (TSP), ACO is straightforward to apply because these problems are already are defined as an optimization on a graph. For our traffic optimization problem, we can clearly evaluate performance of solutions via simulation just like in PSO, but otherwise how to apply ACO to this problem is not straightforward. We describe our novel approach for solving this below.

Suppose there are $N = n^2$ traffic lights, each which can take on one of $m = 64$ possible settings. We allow each of the four main states of a standard traffic light cycle to last for one of two time periods: $t_1 = 15$, $t_2 = 30$. We also allow any of these four states to be the starting position in the cycle at time $t = 0$ in the simulation. This gives $4 * 2^4 = 64$ possible settings for each intersection.

[10] Hu.
[11] AG12.

Define these (intersection, traffic light setting) pairs as the Nm solution components; that is, the edges that ACO uses to build solutions and updates the pheromone values of. All pheromone values e_{N_i,m_j} are initialized to 1.0. To construct a solution, we start with an empty solution and choose among the Nm paths each with probability proportional to the pheromone value: $\propto e_{N_i,m_i}$. Note that this corresponds to setting $a = 1, b = 0$ in the general ACO probability rule (no heuristic data is used). Suppose we choose (N_0, m_0); then we set intersection N_0 to follow traffic light setting m_0. We next choose among $(N-1)m$ "legal" paths, because there are only $(N-1)$ intersections that still need settings. We continue to construct our solution in this manner until all intersections have a defined setting, and then we can evaluate our solution's performance. Each iteration consists of a batch of m such solutions.

This approach differs from standard ACO in that we don't have an explicit construction graph in which we enumerate all the vertices and edges in the graph. A direct application of ACO would require vertices to be all possibilities of which intersections have been assigned settings set already, and what those settings are. This amounts to $\sum_{i=0}^{N-1} \binom{N}{i} \cdot i^m$ total vertices, and exponentially many edges between them. Instead, we use "virtual vertices" that represent each possibility of which intersections have been assigned settings set already, and what those settings are. Each of these vertices have degree Nm, where the same Nm edges (solution components) are shared across all the vertices, and so we look up and update Nm communal pheromone values.

For our pheromone update, we use a novel approach that compares the solution performances within a batch to each other, as opposed to using some fixed function. There are two components to the pheromone update that occur after each batch of m solutions. First, we apply an evaporation update to all Nm edges, which is standard in the ACO framework: $e_{N_i,m_j} := (1-p)v_{N_i,m_j}$. For the second component of the pheromone update, rank the m solutions in order of performance: $f(s_1^*), ..., f(s_m^*)$. Then the additional pheromone update is:

$$C(s_r^*) = (k+1-r)\frac{2p \cdot N \cdot m}{k(k+1)}, [1 \leq r \leq k]$$

$$C(s_r^*) = 0, [k < r \leq m]$$

$$e_{N_i,m_j} := e_{N_i,m_j} + \sum_{r=1}^{m} I[(N_i, m_j) \in s_r^*] \frac{C(s_r^*)}{N}$$

This ensures that $\sum e_{N_i,m_i} = Nm$ after each round, since the evaporation stage decreases $\sum e_{N_i,m_i}$ by $p \cdot N \cdot m$, but then we redistribute this quantity of pheromone based on the relative performance of the m solutions in the batch. We split this pheromone into $\frac{k(k+1)}{2}$ equally sized pieces, then assign k pieces to the best solution, $k-1$ pieces the second best,...,1 piece to the kth best solution. Then, each solution (which defines one of m settings for each of N intersections) distributes this pheromone equally to the N edges that constitute the solution. In line with the ACO framework, edges used in high-performing solutions will gather more pheromone and will be more likely to be included in solutions in subsequent batches. Full ACO psuedocode below.

procedure ACO($n, N, m, T, t_1, t_2, p, k, B$)
    for $i = 1, N$ **do**
        for $j = 1, m$ **do**
            $e_{N_i,m_j} = 1.0$
    for $b = 1, B$ **do**
        for $q = 1, m$ **do**
            Start new path s_q
            for $v = 1, N$ **do**
                Pick $(N_i, m_j) \propto e_{N_i,m_j}$ where N_i not yet in s_q
                Append (N_i, m_j) to path s_q
            $c(t_1, t_2, s_q) =$ Convert s_q into traffic light setting encoding
            $f(s_q) =$ Mean travel time according to $SUMO\_simulation(c)$
        $f(s_1^*), ..., f(s_m^*) =$ Sort$(f(s_1), ..., f(s_m))$
        for $r = 1, k$ **do**
            $C(s_r^*) = (k+1-r)\frac{2p \cdot N \cdot m}{k(k+1)}$
        for $r = (k+1), m$ **do**
            $C(s_r^*) = 0.0$
        for $i = 1, N$ **do**
            for $j = 1, M$ **do**
                $e_{N_i,m_j} \leftarrow (1-p)e_{N_i,m_j} + \sum_{r=1}^{m} I[(N_i, m_j) \in s_r^*] \frac{C(s_r^*)}{N}$

21: **if** $f(s_1^*) < best\_time$ **then**
22: $best\_time \leftarrow f(s_1^*)$
23: $best\_setting \leftarrow s_1^*$
24: **return** $best\_time, best\_setting$

6 Experiments

6.1 Simulator and Traffic Map Details

For our experiments we use the Simulator of Urban Mobility (SUMO)[12]. SUMO is a free and open-source traffic simulator suite with significant modeling power. Despite its flexibility, to be able to consistently recreate experiments and for fair comparison, we fix our simulations as follows. We assume an $n \times n$ grid of equally spaced four-way intersections, each of which is a traffic light. In order to make each traffic light a four-way intersection, we also include boundary nodes on each side of the grid, for a total of $n^2 + 4n$ nodes. Each traffic light cycles between eight states, designed to resemble the cycle for a real-world traffic light. In essence there are four states, which we double depending on which direction (north-south or east-west) has green lights:

1. green light with no left turn arrow, i.e. left-turning cars yield to straight-moving traffic
2. yellow light
3. left turn arrow only
4. yellow turn arrow only

For vehicles, at each time step $t \in [1, 3600]$ the number of vehicles spawned is distributed $Binom(n = 10, p = 0.1)$. All vehicles have the same acceleration rates, max velocities, vehicle size, etc., which we leave as the default. However, each vehicle is assigned a travel route determined in the following way. First, we assign each road in the network a weight distributed $Unif[0, 1]$, which we interpret as a measure of how busy the road is. Then, for each vehicle, we choose a starting location from among the $n^2 + 4n$ nodes in the network. For any vehicles spawned at the $4n$ boundary nodes, the first move in the path is moving to the adjacent four-way intersection. Next, the vehicle is assigned one of NW, NE, SW, SE to represent the two of the four cardinal directions it will move in. To determine a route, the vehicle selects one of the two assigned directions to travel in proportional to the edge weights and adds this to its route. For example, if a vehicle is assigned NW, and the road going north has weight 0.75 and the road going west has weight 0.5, then with probability $\frac{0.75}{0.75+0.5} = 0.6$ the northbound edge is added to the vehicle's path. After each edge is added to the vehicle's route, the path is terminated with probability $\frac{1}{2n}$, so the average path length is roughly $\frac{n}{2}$. We also terminate a vehicle's path if it reaches a boundary node.

Our evaluation metric is the mean time for a vehicle to complete its trip. This value is computed and output by SUMO.

6.2 Experimental Setup

For all our experiments, we use a 5×5 grid for our traffic layout. We fix the edge weights, vehicle spawns, and vehicle routes using a random seed. We also set all phases with yellow lights to be 5 seconds, to mimic real world behavior.

For PSO, we experiment with various bounds on the search space, as well as varying with the number of particles and parameters c_0, c_1. We run all simulations for 500 iterations, use 10 particles for exploration, and cap velocity so that each element has absolute value at most 10. For baselines, we compare against random search for a comparable number of iterations (number of particles * number of iterations) as well as setting all the lights to the same duration.

For ACO, we used a batch size of 64 ants for each iteration and ran most simulations for 75 iterations (each of which took about 8 hours). We experimented with different values for evaporation parameter p, pheromone update parameter k, number of iterations run, and time length of the traffic light settings t_1, t_2. Unless otherwise specified below, parameter values are $p = 0.05, k = \frac{m}{2}, t_1 = 15, t_2 = 30$. Note that when $p = 0.00$, then the result is a random search baseline.

We present our results in Table 1.

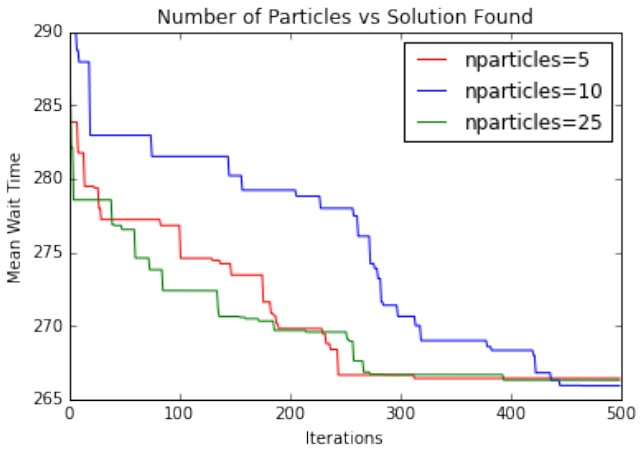
7 Results and Discussion

Of the baselines, we see that random search over the solution space does fairly well, but setting all phases to some constant value does not do particularly well, which makes sense because some roads have more traffic than others, and thus intuitively should receive longer green light periods. Both our methods are able to obtain some performance gain over the baselines.

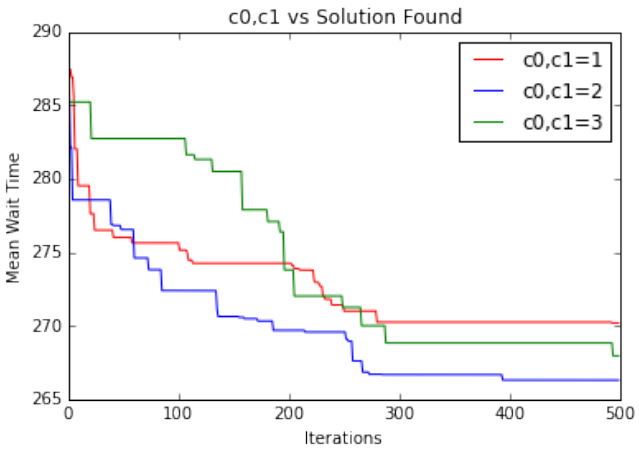
[12] Kra+02.

Representation	Mean Wait Time
ALL NON-YELLOW PHASES 5S	321.44
ALL NON-YELLOW PHASES 10S	291.08
ALL NON-YELLOW PHASES 15S	281.27
ALL NON-YELLOW PHASES 20S	286.18
PSO RANDOM SEARCH	276.86
PSO ON $[5, 30]$	**265.92**
PSO ON $[5, 20]$	267.26
PSO ON $[10, 25]$	269.91
PSO ON $[5, 30]$ + ROUNDING TO NEAREST 5	268.38
PSO ON $[5, 30]$ + STARTING PHASES	275.84
PSO ON $[15, 30]$ + ROUNDING TO NEAREST 15	285.92
PSO ON $[15, 30]$ + ROUNDING TO NEAREST 15 + STARTING PHASE	283.15
ACO RANDOM SEARCH	271.72
ACO WITH $p = 0.05$	264.71
ACO WITH $p = 0.1$	260.41
ACO WITH $p = 0.5$	256.39
ACO WITH $k = m$	267.56
ACO WITH $k = m/4$	261.71
ACO WITH 390 ITERATIONS	252.88
ACO WITH 150 ITERATIONS, $t_1 = 10, t_2 = 20$	**252.18**

Table 1: Best mean wait times for methods and different parameters. For PSO, we use the interval $[x, y]$ to denote that we set $t_{min} = x, t_{max} = y$. "Starting phase" denotes that we did not fix the initial starting phase of each light and made it a parameter to be learned. Rounding to the nearest X means that we rounded each component of the solution to the nearest multiple of $X \in [t_{min}, t_{max}]$.

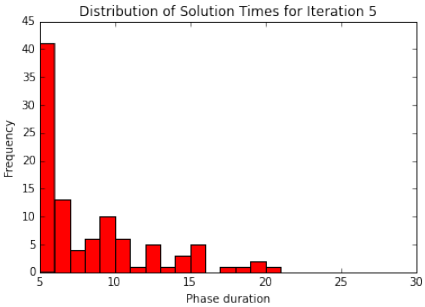


(a) Performance for varying number of particles.

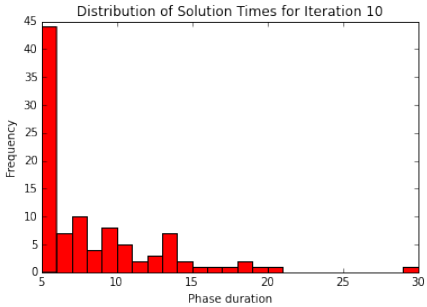


(b) Performance for varying c_0, c_1.

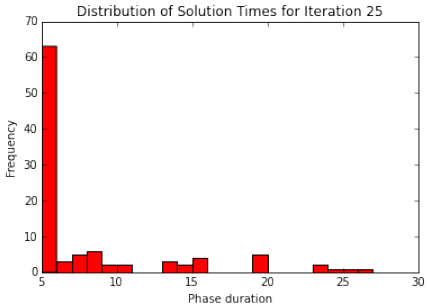
Figure 1: Mean wait time over iterations for different parameters for PSO.



(a) Distribution for iteration 48.



(b) Distribution for iteration 113.



(c) Distribution for iteration 394.

Figure 2: Distribution of components of globally optimal solution over time for PSO.

7.1 Analysis of PSO Results

For PSO, we chose $[5, 30]$ as our initial search space because we found that any larger upper bound would cause accidents and significantly raise wait time. With this assumption, we experimented with different numbers of particles and values for parameters c_0, c_1, as shown in Figures 1a and 1b respectively.

We found that the number of particles actually did not matter significantly in the long run, as all of our experiments eventually converged to roughly the same value. The speed at which a particular iteration converged was dependent on the number of particles, with the highest number of particles converging fastest. Interesting, the algorithm with 10 particles converged slower than with 5 particles, but we suspect that is because we only had enough time to run each algorithm once, and that with more repetitions, the 10 particle algorithm would be faster. Similarly, we found that the optimal settings for c_0 and c_1 to be $c_0 = c_1 = 2$. For the remaining experiments, then, we fixed the number of particles to be 10 and $c_0 = c_1 = 2$.

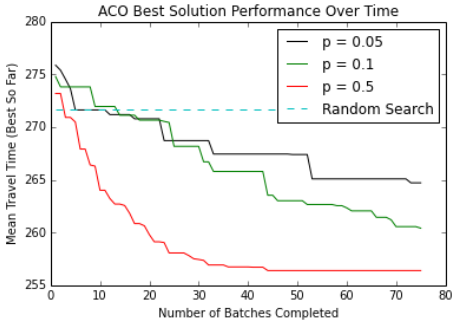
We then investigated the optimal solutions found by PSO. Interestingly, we noted that the algorithm tended to push most durations toward $t_{min} = 5$, so much so that over half of the components in the final solution were set to t_{min}, as can be seen in Figure 2.

With this analysis in mind, we adjusted our experiments in a number of ways. First, we added the simple baseline of setting all parameters to be a constant value. We especially wanted to see if setting all parameters to be 5 would be optimal, which the results show is not. Thus, we could conclude that the phases and lights which were not set to the minimum time were actual improvements learned by the algorithm. Second, based off the fact that the algorithm was rarely assigning higher values in the search space, we repeated the experiment with a narrowed search space to see if the algorithm could learn a better solution faster, as the total search space had shrunk. Third, we observed that the distribution of components was somewhat clustering around particular values, so we experimented with rounding to the nearest multiple of some number, in order to enforce clustering. Our intuition here was that by forcing the traffic lights duration all to be from some finite set of values, e.g. $\{5, 10, 15, 20, 25, 30\}$, there would would be some synchronization benefit, where adjacent lights would be green for the exact same duration and thereby minimize wait time.

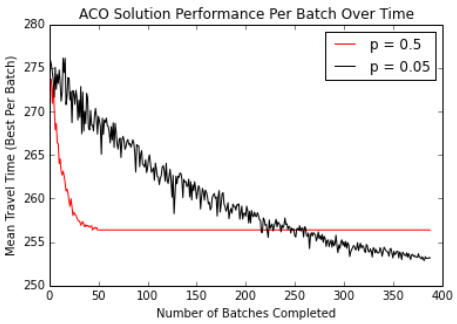
Additionally, we tried expanding the feature representation of a traffic light by parameterizing where in the light cycle each traffic light started, using the aforementioned rounding technique. Our hope was that giving this extra degree of freedom to the traffic lights would lead to performance gains. We note that if we restrict the search space to be $[15, 30]$ and rounded to the nearest multiple of 15 and learned the starting phase, then this formulation is equivalent to the one used in our ACO experiments.

Despite our intuitions, the results show that these extensions did not lead to a better solution, and actually in some cases made it worse. Surprisingly, by reducing the search space, we made the solutions found slightly worse. One possible explanation is that some roads were indeed so heavily traveled that they should have had a green light significantly longer than the perpendicular direction. Rounding to the nearest multiple of some number and learning the starting phase significantly made performance worse. For the former, we speculate that this is because the extension of PSO to discrete values is not particularly fitting. For the latter, we note that the solution found without searching for an initial phase, i.e. setting all lights to start at the same phase, is a subset of the solutions where the initial phases are learned. Thus, ideally learning the initial phase should lead to a solution that is as good as not learning the initial phase. One explanation for why we do not see this is that adding feature representations significantly increases the search space, so much so that it takes longer for the algorithm to find a solution as good as in a smaller search space.

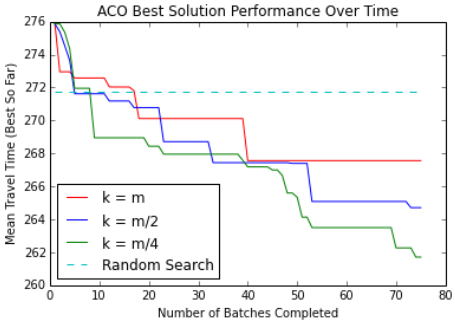
Especially interesting is that our PSO algorithm was significantly outperformed by our ACO formulation both when PSO was less constrained (only searching for phase durations) and when PSO was performing the same search (either 15 or 30 for each phase). Possible explanations for this include those previously mentioned: there is some sort of synchronization benefit from having phases only lasting two possible values, PSO does not extend well to discrete values. Another possible explanation is that the PSO wasn't as strongly encouraged to exploit the globally optimal solution. In ACO, similarly to reinforcement learning, the agents explicitly receive a reward that encourages using the optimal solution and discourages other solutions. On the other hand, in PSO, the encouragement to explore near the globally optimal solution is via the velocity vector that moves an agent toward the globally optimal solution, but also towards local optima. Thus, it could be that PSO does not implement its hueristic of exploring near the global optimum strongly enough. ACO, with its similarity to reinforcement learning, a popular approach for traffic light optimization, may be better suited for this task.



(a) With higher evaporation rates, ACO converges faster.

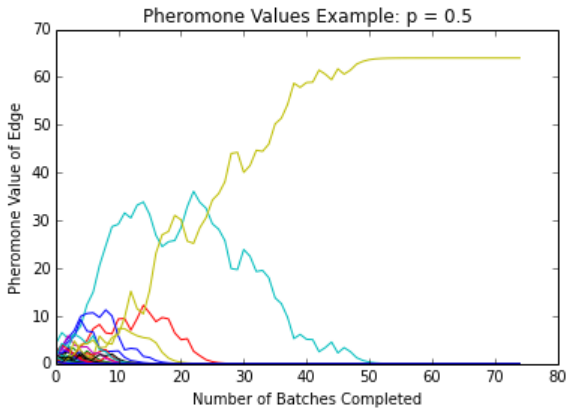


(b) Lower evaporation rates yield better performance after sufficiently many iterations.

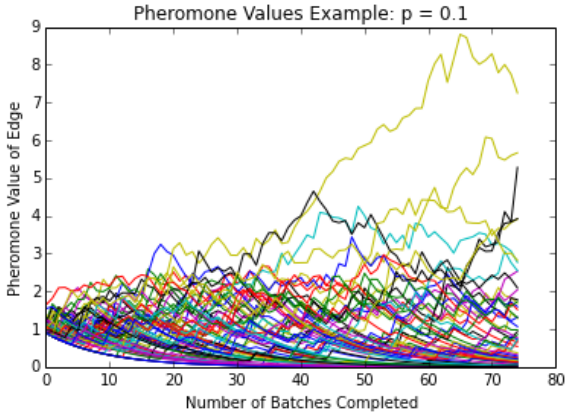


(c) A more generous cutoff for how good a candidate solution's performance must be to receive additional pheromone leads to slower convergence and worse results after 75 iterations.

Figure 3: Changing the evaporation parameter p and pheromone update parameter k in ACO runs. We see a tradeoff between speed of convergence and performance of the ultimate optimal solution found.



(a) With high evaporation rate, all but one pheromone value goes to 0 for each intersection. Once this occurs, ACO cannot find any more optimal solutions.



(b) A lower evaporation rate yields better balance between edges, allowing for continued exploration of optimal solutions by ACO.

Figure 4: Example of the pheromone values for the $m = 64$ edges corresponding to a single intersection over time.

7.2 Analysis of ACO Results

Upon running ACO for 75 batches, we see that higher pheromone evaporation values p appear to perform better than lower values. (Figure 3a) All cases perform significantly better than the baseline random search, shown as a horizontal line at 271.716 (best performance by random search after 75 batches). However, upon closer inspection, we see that higher values of p simply lead to faster convergence, not better performance. This is due to higher values of p leading to more exploitation and less exploration of the search space. We see evidence of this in Figure 3b, which shows that upon running more iterations, we get that the $p = 0.05$ run eventually eclipses the $p = 0.50$ run in performance.

This idea is further confirmed by Figure 4a and Figure 4b, which show examples of pheromone values for an intersection after 75 iterations for the $p = 0.5$ and $p = 0.1$ cases, respectively. The $p = 0.5$ case exploits the best pheromone paths so extensively that the pheromone values have become 0 and 1, and thus there is no more randomness left in the ACO algorithm. Hence, regardless of how many iterations are run, the $p = 0.5$ case will not find any better solutions. On the contrary, there is much better balance in the $p = 0.1$ case, with poor edges having pheromone values fall to 0 but the better edges all maintaining some probabilities of being selected for future candidate solutions. Hence, over time, using lower evaporation rates will lead to better and better solutions. The tradeoff is that smaller values of p will also lead to slower and slower convergence, so finding good solutions will take many more iterations.

We see a similar exploration vs. exploitation tradeoff when varying k, which determines how many of the $m = 64$ candidate solutions in each batch are used to update pheromone values (ex. $k = m/4$ means that additional pheromone is only distributed across edges of the top 16 of the 64 candidate solutions). (Figure 3c)

Finally, noting that the choice of $t_1 = 15, t_2 = 30$ is relatively arbitrary and in light of our findings that shorter time intervals for the traffic light cycle settings seemed more optimal, we completed a run using $t_1 = 10, t_2 = 20$. This yielded performance better than our best run while using $t_1 = 15, t_2 = 30$. Under $t_1 = 10, t_2 = 20$, the algorithm found a solution

with mean travel time of 252.18 within just 150 iterations. In comparison, the best solution found using $t_1 = 15, t_2 = 30$ was 252.88 after 390 iterations, and the best mean travel time was only 258.26 after 150 iterations.

Ultimately, the optimal solution found with ACO achieved a 252.18 mean travel time, a 7% improvement compared to baseline of 271.72. However, even better performance is almost certainly possible; there was no sign that the optimal mean travel time would stop decreasing in the $t_1 = 10, t_2 = 20$ case, and the 150 iteration cutoff was arbitrary. (In contrast, Figure 3b shows diminishing returns to additional iterations in the $t_1 = 15, t_2 = 30$ case.) Further optimizing parameters such as t_1, t_2 and others would also yield even more optimal solutions. We were limited by the long computational time required by SUMO simulations, since running just 75 iterations takes 8 hours. Still, these results demonstrate promise in this ACO approach.

7.3 Ideas for Further Work

For Particle Swarm Optimization (PSO), though our extensions were not entirely successful, we believe that there is still significant room for experimentation here. For example, for PSO, we wanted to explore using non-constant values for c_0, c_1, with the intuition being that we could control the degree to which the particles initially explore by setting c_0, c_1 low, then increasing them over time to exploit. Some type of simulated annealing approach that gradually increased c_0 and c_1 over time might see promising results. Another possible direction for future work would be exploring other feature representations, particularly ones that could associate adjacent traffic lights to learn to synchronize them. Today, many modern traffic light systems use time of day and traffic volume features to inform their behavior; incorporating such features could also lead to performance gains in our method.

For Ant Colony Optimization (ACO), one possible idea is as follows. When we have these Nm solution components, we are implicitly assuming that regardless of what settings we assigned to other intersections already, there is some same inherent quality associated with next assigning intersection n_i to some setting m_j. This doesn't seem realistic because the settings of intersections around it should impact what setting our current intersection should be assigned. Our above approach allows adjacent intersections to impact each other only indirectly, not directly. To solve this, we could use $Nm(m+1)^4$ solution components, representing which of N intersections we're currently setting, which of m settings we set it to, and additionally what the settings are for each of the four adjacent intersections (including the case of not being set yet). We might also impose some deterministic ordering on when the intersections are set, such that adjacent intersections are set consecutively. We could then just sample different orderings over the intersections. We anticipate this idea would fare even better than our ACO implementation. However, it was not feasible for us to experiment with this due to computational limitations. $Nm(m+1)^4$ is much larger than Nm, so many more SUMO simulations would be needed for any meaningful convergence in edge pheromone values.

Above all, we found that the limiting factor in our experiments was the time per simulation. Finding better hardware or more computing power to run more simulations would greatly expedite any further work on this topic.

8 Conclusion

In this work, we applied swarm optimization methods to the problem of traffic light optimization. We demonstrated that our approaches were able to make improvements in mean waiting time for the system over baseline methods. Importantly, our methods do not rely on any assumptions made about the network we were optimizing for, and thus can be extended to arbitrary traffic conditions and assumptions.

Interestingly, PSO, which we had originally believed would be a fruitful approach for this problem, did not have huge performance gains over the baselines. In fact, our novel application of ACO to this problem significantly outperformed both the baselines and our PSO method, despite having more constraints on what the time intervals within a traffic light cycle could be, suggesting the validity of ACO for non-graph problems. Application of this version of ACO, in which many vertices share a set of edges (Nm such edges in our case) and a set of values (pheromone values in our case) is continually updated using data to parameterize optimal solutions, shares characteristics with reinforcement learning. Indeed, it could prove fruitful to use such an approach on other problems traditionally solved using algorithms like Q-learning or value/policy iteration.

Our code with more complete implementation details and design decisions can be found at
https://github.com/W4ngatang/TrafficSwarmOptimization.

References

[AG12] Hazem Ahmed and Janice Glasgow. "Swarm Intelligence: Concepts, Models and Applications". In: Queen's University, School of Computing. 2012, pp. 1–38 (cit. on p. 3).

[DD98] Bart De Schutter and Bart De Moor. "Optimal traffic light control for a single intersection". In: *European Journal of Control* 4.3 (1998), pp. 260–276 (cit. on p. 1).

[ES01] Russell C Eberhart and Yuhui Shi. "Particle swarm optimization: developments, applications and resources". In: *evolutionary computation, 2001. Proceedings of the 2001 Congress on*. Vol. 1. IEEE. 2001, pp. 81–86 (cit. on p. 1).

[GOA13] Jose Garcia-Nieto, Ana Carolina Olivera, and Enrique Alba. "Optimal cycle program of traffic lights with particle swarm optimization". In: *Evolutionary Computation, IEEE Transactions on* 17.6 (2013), pp. 823–839 (cit. on p. 2).

[Hu] Xiaohui Hu. "Particle Swarm Optimization: Tutorial". In: *swarmintelligence.org/tutorials.php* (cit. on p. 3).

[KB09] Sofiene Kachroudi and Neila Bhouri. "A multimodal traffic responsive strategy using particle swarm optimization". In: *Control in transportation systems*. 2009, pp. 531–537 (cit. on p. 2).

[Ken11] James Kennedy. "Particle swarm optimization". In: *Encyclopedia of machine learning*. Springer, 2011, pp. 760–766 (cit. on p. 2).

[Kra+02] Daniel Krajzewicz et al. "SUMO (Simulation of Urban MObility)-an open-source traffic simulation". In: *Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002)*. 2002, pp. 183–187 (cit. on p. 5).

[SGR04] Javier J Sanchez, Manuel Galan, and Enrique Rubio. "Genetic algorithms and cellular automata: A new architecture for traffic light cycles optimization". In: *Evolutionary Computation, 2004. CEC2004. Congress on*. Vol. 2. IEEE. 2004, pp. 1668–1674 (cit. on p. 1).

[Ste+05] Merlijn Steingrover et al. "Reinforcement Learning of Traffic Light Controllers Adapting to Traffic Congestion." In: *BNAIC*. Citeseer. 2005, pp. 216–223 (cit. on p. 1).

[Wie+04] Marco Wiering et al. "Simulation and optimization of traffic in a city". In: *Intelligent Vehicles Symposium, 2004 IEEE*. IEEE. 2004, pp. 453–458 (cit. on p. 1).