Smart Parking

David Bain, Patrick Xu, Will McConnell

Introduction

For our project, we chose to analyze "smart" parking as it compares to "dumb" parking. There are numerous "smart" parking systems on the market currently, but we chose to model ours after a system like "Streetline," which uses sensors to relay information to a user about open parking spots near a location of interest.[1] Using this system, a user is efficiently directed towards an open spot. On the other hand, "dumb" parking is in place today, in which a person searches nearby city blocks until they find parking near their location.

Given the explosion of the Internet of Things, it is likely that systems like smart parking will become more widespread as "smart" city infrastructures are implemented. Therefore, we sought to develop a simulation that could quantitatively determine the impact of smart parking by measuring average time spent looking for parking and average distance from the parking spot to the destination. With this data, we could predict the effect of different parking systems on important variables like traffic congestion and carbon emissions. Some experts predict that thirty percent of city traffic is due to drivers looking for parking.[2] Therefore, optimizing parking would have an immediate and powerful impact on citizens of a "smart city."

Methods

Using the SimPy discrete-event simulation framework, we developed a simulation to analyze and compare smart and dumb parking algorithms. The project, hosted at https://github.com/patxu/parking_sim, has four main parts: map generation and representation, simulation, and graphics, testing.

Map Generation and Representation:

A city is represented using the following classes:

- `Coord` - Represents an (x,y) coordinate, creating a grid upon which all other classes operate.
- `ParkingSpot` - Represents an individual parking spot and maintains the state of the parking spot.
- `RoadSection` - Represents a discrete section of road at a coordinate point and contains a left and right `ParkingSpot`. Holds boolean values indicating whether or not it is an intersection and crossable for parking, as well as a directionality indicating whether or not the road is vertical (North) or horizontal (East).
- `Road` - Represents a linear road. Has a unique id, list of road sections, directionality, a fixed coordinate point (e.g. `x=1`) and a min and max value of the road (e.g. $0 \le y \le 100$).

- `RoadMap` - Represents the city as a whole. Contains a list of roads as well as a graph of roads where each node is a `Road` and each edge is an intersection of two roads

A more static representation of a city can be found in an xml representation. The xml closely resembles the in-memory form, and can be used to generate and load cities into memory. Check out `generate.py` and `roadmap.py` for code relating to this aspect of our project.

Simulation:

The simulation runs in the SimPy framework. The main actor in the simulation is the `Car` class. The basic flow of the simulation is as follows:

1. Each car is initialized and its behavior set (either “smart” or “dumb”)
2. Each car is randomly placed on a road in the city
3. Each car receives a specified number of destinations chosen at random
4. At each time step:
    a. If it is parked, release the parking spot
    b. If no more destinations
        i. Begin random movement behavior
    c. If wants to park and parking spot available
        i. Park
        ii. Pop destination from stack of destinations
    d. Else
        i. execute movement behavior

There are three movement behaviors: Random, Smart, Dumb. Random behavior is as expected- the car makes a random driving choice at intersections, with a u-turn being the least desirable choice. Smart behavior requests the closest open parking space to its destination and heads in that direction. It updates the closest open spot at every step of the simulation. It only parks when it is at the closest assigned open spot, and will reroute if their assigned spot is taken, either by another smart `Car` or a dumb `Car`. Dumb parking cars head towards their destination. Once in a certain radius (for the simulation, a distance of 20 units) it begins looking for a parking space. If the car gets to the destination and has not found a parking space, it begins a circling pattern where it circles each of the four blocks around the destination twice before moving onto the next block. It does this until it finds a parking space. The simulation runs until more than 97% of the cars have reached and parked at all of their destinations. We choose 97% rather than a 100% to ensure the simulation doesn’t hang even if there is an unpredicted corner case in which car gets trapped in a loop (we don’t think this is the case, but better safe than sorry). Check out `simulation.py` and `run.py` for code relating to this aspect of our project.

Graphics:

We decided to use `cs1lib.py`, the python graphics library provided to CS1 students. Because of how this library is structured, the functions that draw on the canvas must be in the same file as the function that runs the graphics, which, since this function can't accept arguments, must also be in the same file as the actual simulation. For this reason the graphics layer is also found in `run.py`.

Testing:

We tested on four maps which can be found in the `cities/` directory. Each of these maps is of size 100 with 10x10 blocks with varying percentages of open parking spots:

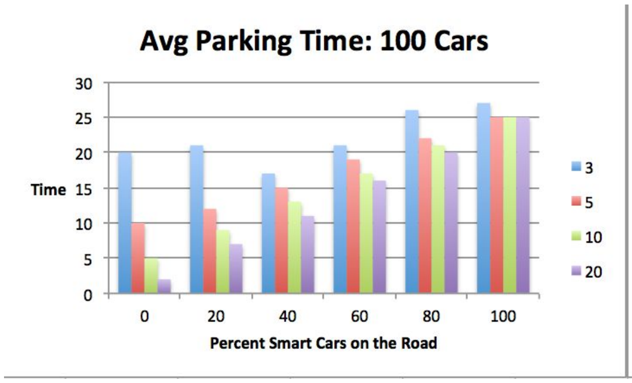
- `percent3_10x10.xml` - 3% parking density
- `percent5_10x10.xml` - 5% parking density
- `percent10_10x10.xml` - 10% parking density
- `percent20_10x10.xml` - 20% parking density

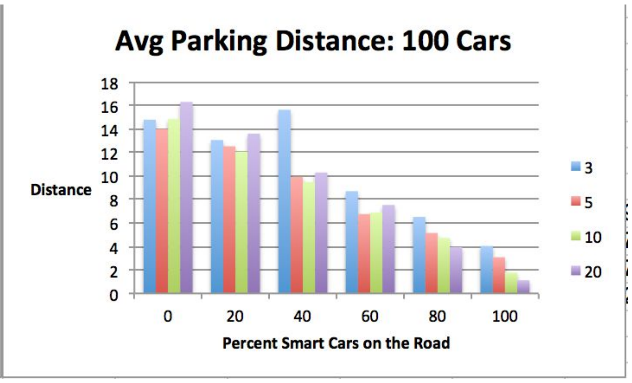
On each of these maps we ran the simulation with 100, 500, 1000, and 2500 cars. For each of these combinations we varied the percentage (100%, 80%, 60%, 40%, 20%, and 0%) of smart cars on the road for a total of 96 simulations.

Because of the large number of tests, testing was automated using the `Makefile`. Run `make generate` to generate the appropriate city files. Run `make smart<% Smart Cars>` (e.g `make smart60`) to run the appropriate set of tests. View the README for more detailed instructions.

Results

The simulation results are stored in `logs/`, with the relevant information at the bottom of each file. The data can also be viewed online at http://tinyurl.com/cs69parking.





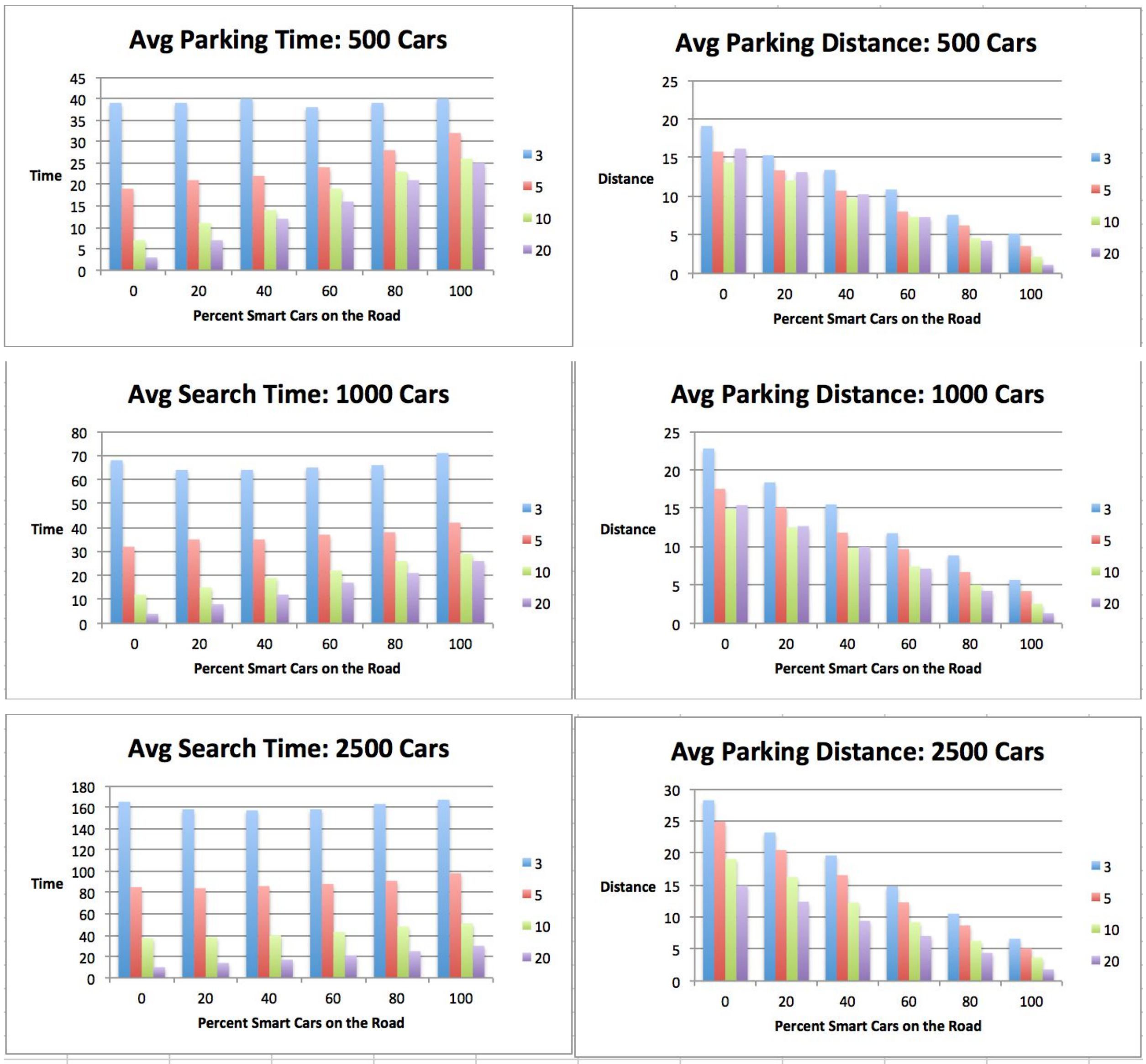


Figure 1: Each bar graph represents the average time or distance given the percentage of smart cars on the road and the density of available parking spots.

Our model predicts that increasing the percentage of cars using our smart parking algorithm will not decrease average search time. However, it does drastically decrease the distance between the parking spot and the destination (Figure 1). Average search time is relatively constant for each varying percentage of smart cars on the road, and actually may increase with a higher percentage of smart drivers when there are fewer cars (e.g. 100 cars) on the road (Figure 1).

Discussion

An interesting comparison between “smart” parking and “dumb” parking is the fact that smart parking can actually lead to users spending more time searching for parking, especially when there are fewer cars on the road. This is likely caused by our relatively simply parking distribution algorithm. The system will direct each car towards the parking spot closest to their destination. However, if this parking spot gets taken by another car, then the car will have to recalculate their route to another free parking spot. Currently, there is no way for a user to “reserve” a parking spot, so their desired parking spot is always in danger of being taken by another smart or dumb car.

There are several approaches which can optimize the smart parking algorithm to prevent these “races.” The distribution of parking spots could be weighted according to the number of nearby spots, or routing only one smart car to one parking spot. However, to guarantee that a “dumb” driver will not take the spot would require a whole separate, complex system that also accounts for “dumb” cars.

The increased time-to-park of our smart parking algorithm can be observed most directly when there are fewer cars on the road. “Dumb” parkers will simply grab the first available spot, while “smart” parkers will continue until they reach their assigned spot (and thus have extremely low distance-to-destination). Keep in mind that in situations when parking is abundant, an actual human driver may choose to skip the first few open spots with the knowledge that there are likely more ahead. Therefore, a smart parking system may be of little benefit when parking is abundant, but could certainly guarantee better parking spots to users in a congested city.

Although actual “smart” drivers may deviate from their assigned spots if they pass many unoccupied spots near their destination, our simulation does suggest that smart parking cars tend to be less easily satisfied, and will not park until they reach their desired spot. This may land them a parking spot closer to their destination, but in the process, they may end up jumping from destination to destination if the spot is continually taken. We feel that this theoretical scenario could certainly carry over to the real world.

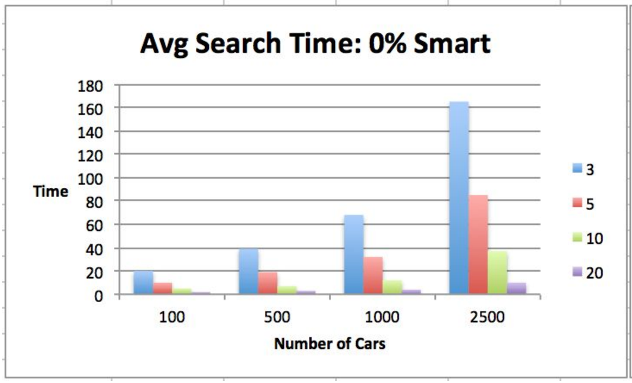
Therefore, smart parking may have a negative impact on congestion and carbon emissions if it leads to users spending more time on the road looking for parking spots. The system could result in more cars on the road at a given time, possibly increasing carbon emission and congestion. Still, as more sophisticated parking algorithms are implemented and tested, smart parking cars may eventually have lower search times and distances to their destination.

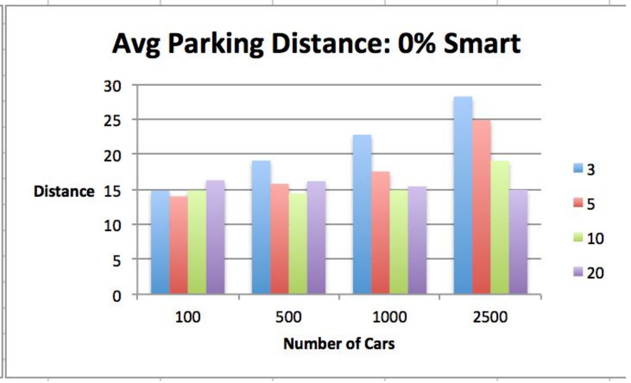
Overall, the chief benefit from this system is guaranteeing that users don’t have to walk as far to their destination in a congested city. This could increase public safety if the user is forced to park in a dangerous neighborhood, but could also be a detriment to public health as it promotes users to be more lazy.

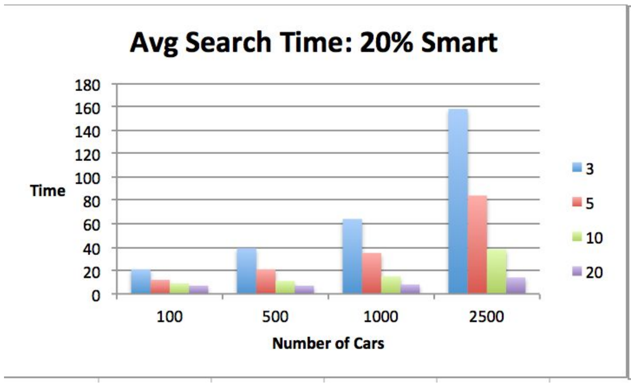
Conclusion

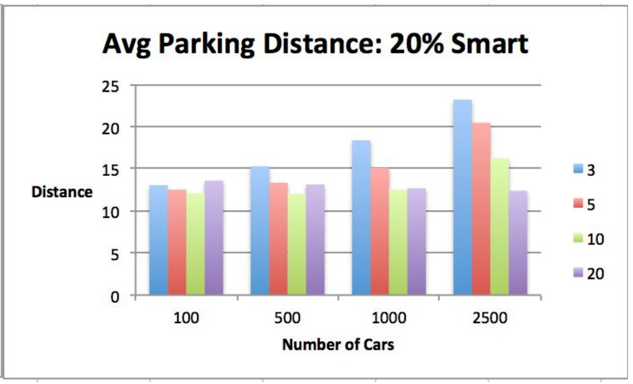
Using the Simpy framework, we were able to develop a program that simulates parking under both a “smart” and “dumb” parking algorithm. Upon comparing the two under various conditions, we found that our smart parking system may not guarantee less time spent on the road, but often guarantees them a better parking spot. Ultimately, our system's impact on a larger considerations like the congestion and carbon emissions is unclear, but possibly detrimental.

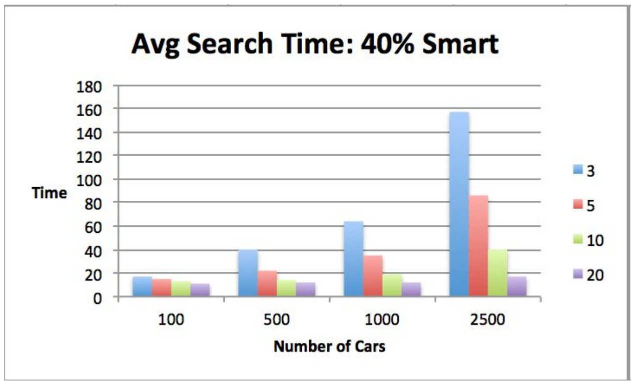
Supplementary Figures

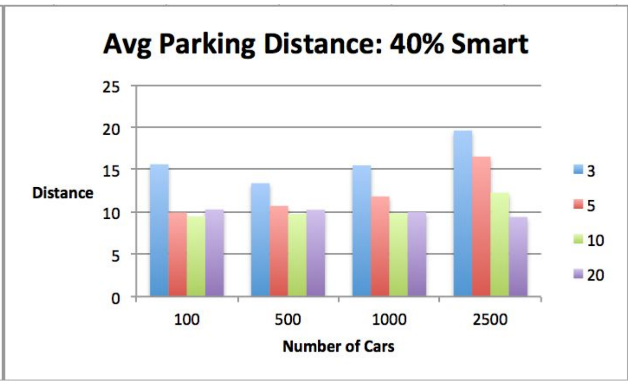












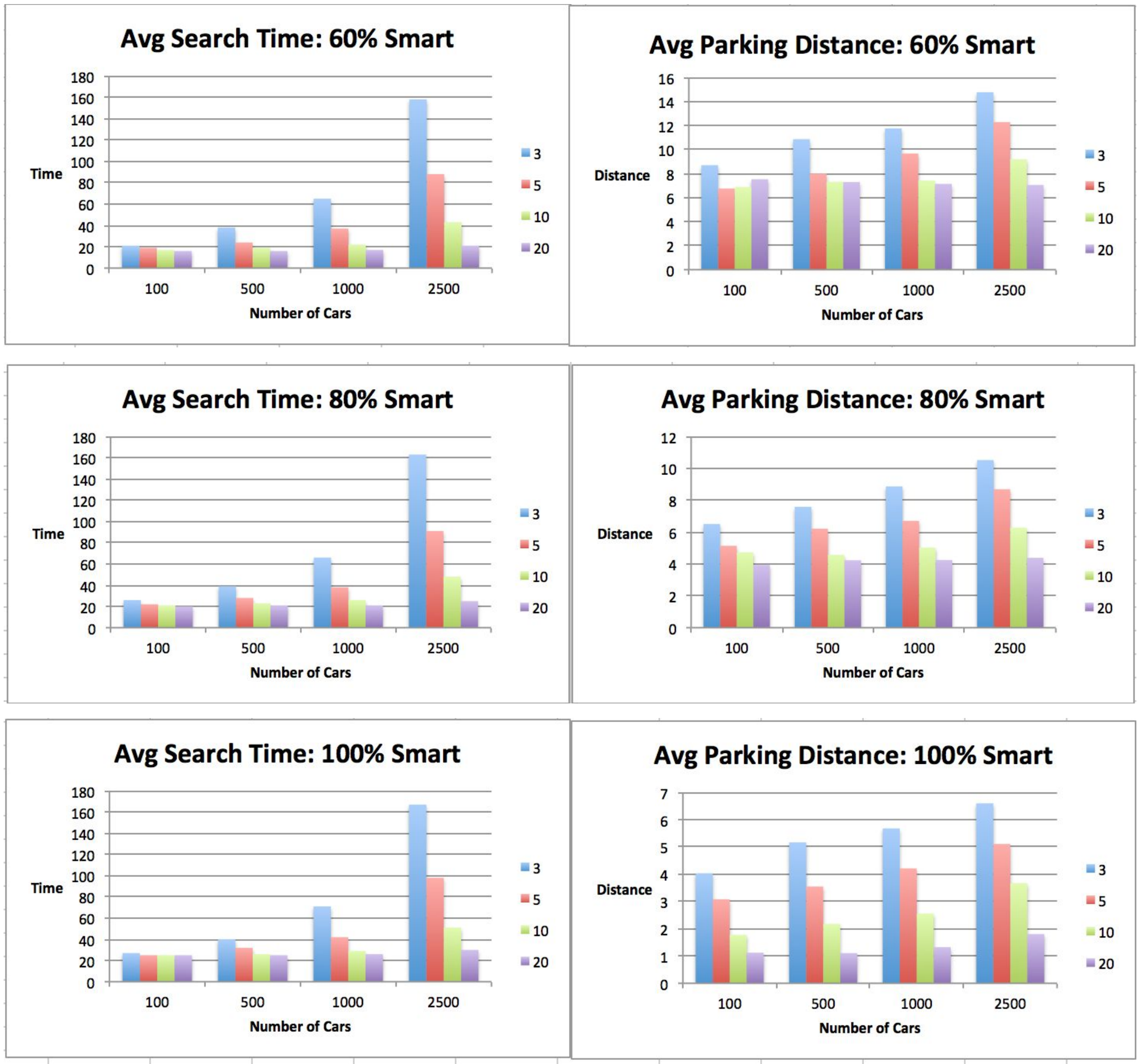


Figure S1: Each bar graph represents the respective average time/distance for each car given the number of cars and density of parking spots (denoted by the different colors, 3%, 5%, 10%, and 20%). As expected, our model predicts that with more cars on the road, average search time and distance will also increase based on more actors competing for fewer resources. Note, however, the vastly reduced average parking distance as the percentage of smart cars increases.

---

[1] http://www.streetline.com/manage-parking/smart-parking-why-choose-streetline/ 8/28/15.

[2] Roberts, Chris. "San Francisco transit agency says drivers seeking parking account for 30 percent of traffic, but data questioned" *San Francisco Examiner*. 9/17/13.