Optimal Routing Algorithm in Swarm Robotic Systems

Moya Chen, James Macdonald
Department of Computer Science
California Institute of Technology
1200 E California Blvd
Pasadena, CA 91125
{mpchen, jmacdona}@caltech.edu

ABSTRACT

As unmanned aerial vehicle (UAV) technology improves, it will become increasingly efficient to use UAVs as part of package delivery networks. However, while the vehicle routing problem (VRP) is well studied, studying routing on a system of drones that can near-instantaneously transmit information about position, held packages, and destination while also able to near-instantaneously recieve new commands is difficult. Unlike simpler versions of the VRP, the routing problem on UAVs that we present is not only dynamic, stochastic, multi-vehicle, and capacitated, but also does not operate with a hub.

This paper gives an introductory analysis of routing algorithms on package delivery systems with UAVs. We present a flexible theoretical model. We then show some saturation limits on a simplified version of the full model using a greedy algorithm.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing protocols; F.2.2 [**Nonnumerical Algorithms and Problems**]: Routing and Layout; I.2.9 [**Robotics**]: Commercial Robots and Applications

General Terms

vehicle routing problem, routing algorithms, traveling salesman problem

1. INTRODUCTION

Unmanned aerial vehicle (UAV)[1] technology has developed significantly in recent years. Modern-day drones are now capable of doing an assortment of complex tasks, tasks as varied as throwing and catching a vertical pole [6][2] to autonomously navigating through a space [7].

With these improvements on the flight ability of UAVs, it is now possible to consider a broader scope of application for these drones. One proposal has been to use these UAVs as part of a transportation network.

There are many reasons why a drone-based transportation network is attractive. Consider the differences between small arial vehicle-based transportation system and existing ones. Unlike large trucks, airplanes, or other vehicles in traditional delivery systems, UAVs are smaller and have more mobility. Because they are capable of autonomous flight, quadcopters can be programmed to fly to some location almost instantaneously, as long as a network connection is available. Since they do not need to travel on roads, quadcopter flight is not inhibited by ground traffic. In these ways, a drone-based network has more flexibility.

An example of a proposal attempting to exploit the benefits of UAV flight include the recently announced "Prime Air" delivery system by Amazon[5]. In Prime Air, claims that it will be able to fullfill orders of its customers in 30 minutes or less, using UAV technology to bring parcels directly to customers' doorsteps.

There have also been other proposed uses for a drone-based delivery system. Unlike trucks, cars, or planes, aerial vehicles do not require expensive infrastructure such as highways or airports. In third world countries where construction of such objects may be cost prohibitive, a UAV network can be used as a way of delivering food, water, medical supplies, and other vital necessities for a much cheaper price. On this front, Matternet (`http://matternet.us/` and similar startups have already begun investigating the requirements for the implementation of such a system, with trials having occurred in both Haiti in the Dominication Republic.

Additionally, as a topic of study, UAV delivery networks have a set of unique conditions not present in many existing networks.

1. Unlike packets in a network, an existence of a node does not guarantee that a package will be able to immediately move through that node. There must also exist a drone to move the package. In this sense, there is some risk of starvation.
2. Trucks, trains, and airplanes generally do not have

[1]In this paper, we use "UAV", "drone", and "quadcopter" interchangably.
[2]While a little unrelated, we recommend the following video: `https://www.youtube.com/watch?v=pp89tTDxXuI`.

their routes changed once they have been set on some path. Because of the ability for drones to communicate via the Internet our model included the ability to change route any time a node is reached.

3. We feel that loading and unloading with drones will be cheap. Thus, our model included being able to drop off a package at some node, even if it is not the destination node.

2. MODEL

In this paper, we present two models. The first is the full model, describing an ideal drone package system incorporating all of the abstractions needed to capture all of the detail of the drone package delivery which we present here. The second is a restricted model that we used for simulation. We will present that model with the simulation.

2.1 Full Model Description

1. A global clock with discrete time t.
2. A graph G with nodes n_i and edges $e_i = n_s, n_t$. Edges have an edge weight d_{e_i}
3. Packages $p_{n_s,n_d,t}$. These packages have start node n_s, destination node n_d, and enter the system at time t. Associated with these packages is a probability function $\Pr(n_s, n_d, t)$ to *stochastically* determine arrival times.
4. A set of robots, R, where each r_i can move along the graph. Each robot is initialized at some node n_i. These robots have some weight capacity w_{r_i}, making our model *capacitated*.

At each t, it is the systems job to decide 1) whether or not to move a robot, given the probability of a new package approaching and the arrival of other packages/robots 2) if so, to which node and 3) while picking up/dropping off which packages such that it minimizes the cost function $C(P)$, which is a cost function applied to all packages in the system. Because the command from the system are sent to each drone at every time step, the system that we present is *dynamic*.

2.2 Full Model Justification

We design our model as we do for many reasons. Because we assume that we will have real-time communication about drone location while also being able to send quick commands, we want our system to be dynamic. Since real customers have requests that occur somewhat randomly, we have our full model be stochastic. We also assume that our system is hubless – essentially, we invision each node to be a drop off/pick off point for packages that includes something like a small set of charging stations such that we do not require a centralized hub. As energy technology grows, we believe this is something feasable to include.

2.3 Basic Model Description

1. A global clock with discrete time t, all timesteps are unit intervals.
2. A graph G with nodes n_i and edges $e_i = n_s, n_t$. Edges have an edge weight of 1.
3. Packages $p_{n_s,n_d,t}$. These packages have start node n_s, destination node n_d, and enter the system at time t. The packages are generated at a start node with a given frequency.
4. A set of robots, R, where each r_i can move along the graph. Each robot is initialized at some node n_i. These robots can carry 1 package (eliminating the capacated nature of the problem) and cross one edge per unit time (move at speed 1).

2.4 Basic Model Justification

Our simulation archetectre can represent the full model; however, we decided that we wanted to study the effects of graph structure and package arrival locations on the wait time for packages and the number of robots required to keep the system clear of packages. In order to do so we eliminated the complexity of varying robot speed and capacity along with the stochastic package arrival. This allowed us to simplify the analysis of the performance of the system. We note that the greedy algorithm we studied did not take the periodic arrival of packages into account.

3. EXISTING WORK

In this section, we discuss some of the research into existing work we did during the exploratory phase of the project. We divide this part into two sections: one for work done prior to the start of the project period and one for after, to illustrate some of the progress that we made in understanding some existing research.

3.1 Initial Research

3.1.1 Vehicle Routing Problems

In transportation, classes of problems suchs as the vehicle routing problem deal with finding the optimal delivery or collection route, given a set of constraints [1]. In many ways, the problems of swarm robot routing are similar to that in vehicle routing problems (VRPs), specifically the capacitated VRP with multiple vehicles.

The capacitated VRP with multiple vehicles has many of the same requirements as our proposed problem. In the capacitated VRP with multiple vehicles, one needs to deliver goods to the recipients using multiple vehicles with specified capacities. The goal is also to minimize route cost. However, since there is no central hub and the routing decisions are made each time step our problem has significant differences from the standard VRPs.

We researched several methods that have been applied to VRPs. One of the most interesting was the Simulated annealing heuristic [2]. Simulated annealing gives each state of the system an energy value depending on the cost of that state. In this case a state would be a potential route and its energy would be the cost. Neighboring states are states with the order of one pair of nodes swapped. During each iteration the algorithm examines neighboring states and transitions as determined by a probability function function $P(e, e_{new}, T)$ of the current state's energy, the new states energy and the Temperature. The function has the property that as T tends to 0 the probability of transition to a state where $e_{new} > e$ tends to 0. In effect the algorithm becomes more greedy as the temperature is reduced,

while begining in a less greedy manner to avoid becoming trapped by a local optimum solution. This method works well when a fairly large amount of computational resources can be devoted to to determining a routing plan that does not need to updated frequently, as in the case of determining the optimal route for the next days package deliveries.

However, in our model robots do not have to return to a central hub to pick up packages and routing decisions can be made whenever a robot arrives at a node. Since examinging the full system to compute an optimal route is expensive, and decisions can be made at each stop it may be optimal to compute only a next step instead of a full route. Thus, this particular ARP model and approprach may not be as appropriate as others.

3.1.2 Graph Properties

In thinking about ways of doing optimal routing on our system, we examined the feasibility of dividing a graph into clusters by minimizing average distance between verticies in the same cluster. This would allow us to reduce the size of the graph to be considered for intra-cluster operations as well as potentially reducing the number of nodes to examine for inter-cluster opperations. Unfortunately we found that a similar problem, partition a graph so as to minimize subgraph diameter, is NP-hard [3].

3.2 In-Project Research

As part of the early weeks of the term, we continued doing research on the dynamic, stochastic form of the VRP. By doing so, we were able to come to models that were much, much closer to the model that we proposed above. While there were still differences, many of the techniques found in these papers seem promising. Most of these approaches choose some method of calculating a traveling salesman like tour of the requests or a portion of a tour and then reculculate periodically or when the sub-tour is complete.

3.2.1 Policies based on TSP

These approaches use a standard TSP tours over a portion of the graph, or a portion of the tour, in order to remain flexible to incomming service requests. Generally the TSP tour is approximated using the Lin-Kerninghan heuristic. The single vehicle divide-and-conquer policy (DC) starts by partitioning the graph; then a TSP tour is computed on the partition with the highest volume of requests. This tour is executed and another tour is cumputed when it is completed and the proess repeats. With multiple vehicles each vehicle is given its own partition the single vehicle policy is implemented within each sub-graph. Another approach computes a full TSP tour and then selectes a portion of it to be cmpleted by each robot. Then after a certain time, or when all tour segments are completed the tours are recomputed[4].

3.2.2 Metaheuristic Approach

Many of these approaches are based on the tabu search heuristic created by Fred Glover in 1986. Tabu search is similar to simulated anealing, however, it allows the process to exclue potential solutions that violate some criteria, called a tabu (as well as solutions that have been tried recently). The metaheuristic maintains a pool of potential routes. When a new request is received it is checked for compatability against all the routes in the pool and the new best candidate is chosen[9].

Some research has also been done applying genetic algorithms to the dynamic vehicle routing problem. Genetic algorithms generate a pool of potential solutions and combine elements of the solutions to contain offspring, which are checked for fitness before themselves breeding new potential solutions. In order to apply this heuristic to a dynamic problem the approach takes a snapshot of the state and breeds paths for the next time slice. One of these solutions is chosen and run until the end of that time slice at which point the process repeats. This approach is similar to the partial TSP method described above, however, it uses genetic algorithms to find a path instead of a TSP solver[11].

4. SIMULATOR

Our simulator archetecture is capable of representing our full model. However, the simulations we ran focussed on the basic model presented above. The cost function we used was simply the amount of time that a package spends in the system. [3]

We wrote our simulator primarily in C++, using the libraries and additional programs mentioned below. We also used some amount of bash and python scripting to aid in generation of output animations and images, which we have also described below.

4.1 Design Overview

The simulator was designed with the goal of making the system as modular as possible.

In order to achieve this, rather than hard coding in any parameter that could be changed (ex. cost function, routing policy used, generation scheme of packages),

This allowed us to implement several robot child classes with differing speeds, capacities, and special routing rules. Each input source of packages is also modular, allowing us to generate packages with different user defined functions.

In order to use these different settings without having to recompile the system every time, we defined a "UAV Input File" format. In this format, we have the following sections:

1. Graph Adjacency List (for describing node structure)
2. Drone Initializer Settings. Must include type of drone (regular, capacitated, package-targeted, etc) as well as starting node for each drone.
3. Package Creator Settings. Each line includes information about node for which package creator is to act upon, destination of package (ex. fixed or random), type of package creator (completely random, random with fixed average, interval, etc) and any extraneous parameters that type of package creator needs.

[3]This is suboptimal in a more general model. However, since we primarily implement the greedy algorithm, it works for most of our purposes. Generally, something hyperlinear would be better since (intuitively, on average) older package should have higher precedence.

4. System Routing Algorithm Type.

Each of these different settings generate a set of different classes which interact, but are otherwise separated from each other. This allows our system to be easily extendable for future use.

4.2 Output

In addition to simulating the simple UAV package delivery system itself, we also integrated together a series of tools to aid us in generating both output video and output graphs.

As seen in 1, we used a variety of additional libraries to aid us in generating our output.

4.2.1 Data Graph Generation

Data graph generation was fairly straightforward. At each timestep, the simulator would write important system status information (ex. number of packages in system, average length of time of packages in system, information about drones, etc, etc) to a set of output files, with one output file for each series of data. After we produced this, we used a simple python script to plot the graphs in the output files.

4.2.2 Animation Generation

Animation generation took a decent amount of background work. For this part, we used the OGDF graph library[4].

At the beginning of program initialization, we would make a copy of our network graph in the program OGDF. During each timestep of simulator update, the system class would take a snapshot of its current status, updating the label of each node in our graph in its corresponding OGDF graph. [5] This graph would then be output to a .gml file.

Once the simulator was done running, we used the program GML2PIC[6] to convert our .gml files to images.

From here, we stiched together the images using the FFMPEG[7] library.

We built a set of bash scrips that would allow us to run all of these operations in one command-line input. This one line input included flags for options relating to input file location, output file location, and various other internal image settings.

4.3 Greedy Algorithm

The greedy algorithm we used in our simulation updates the destination of each robot in a simple manner. We use a globally implemented shortest-all-paths table to calculate where each drone with a destination should go to next.

Upon generation, each drone does not have a destination. Once packages begin entering the system, the oldest package in the system is assigned to some destination-less drone. The destination of that drone is then changed to the current location of that package. Once the drone is at the location of the package, the drone picks the package up, changing its destination to the end point of the package. Once the package's goal node is reached, the package is considered dropped off, and the drone's current destination is set to "none".

4.4 Challenges

In implementing our simulator, we ran into quite a few challenges.

Contrary to what one might think, libraries for implementing dynamic graph updating do not readily exist for C++. There are an assortment of static graph generators, however. For example, the group attempted to use Boost:Graphviz[8], another graph creation library, in order to generate images for our animations, however we ran into issues upon realizing that the library regenerated an image of our graph every single time, including recalculating graph layout. This was something that was not at all appropriate for an animation.

The solution that we settled on, OGDF got around this problem by storing the locations of nodes in our graph in its own graph structure. However, this meant that whenever anything was changed, we had to manually generate labels to interface with OGDF's print format such that the text would appear properly.

Even while using OGDF, we noticed a few limitations. For example, it would not be particularly well suited for a graph with a large number of nodes since the screen would be overly full. Additionally, the layout algorithms we tried never quite seemed to be as optimal as we would have liked them to be.

In hindsight, it perhaps would have been easier to have implemented a dynamic graph generation library from scratch. However, given time constraints, this did not end up occuring.

5. RESULTS

As a baseline, we ran a few tests of the Greedy Algorithm on a variety of graph structures with different types of package generation.

In figures 3-8, we provide graphs of the average number of packages in system given a variety of different graph structures and package generation schemes.

These behave as one would expect. As the number of drones increase, the increase in the number of packages in the system decreases. It is also possible to notice some very clear threasholds in some of the graphs.

It is possible to come up with a fairly simple explaination of why these threasholds are located where they are in some of the graphs. In the line and ring networks, the average package spawn rate is 1 package over the entire system per time step. Thus, the amount of additional distance that

[4] `http://www.ogdf.net`

[5] For why our program behaves this way and challenges involved in getting to that point, see 4.4

[6] `http://www.ogdf.net/doku.php/project:gml2pic`

[7] `http://www.ffmpeg.org`

[8] `http://www.boost.org/doc/libs/1_55_0/libs/graph/doc/write-graphviz.html`

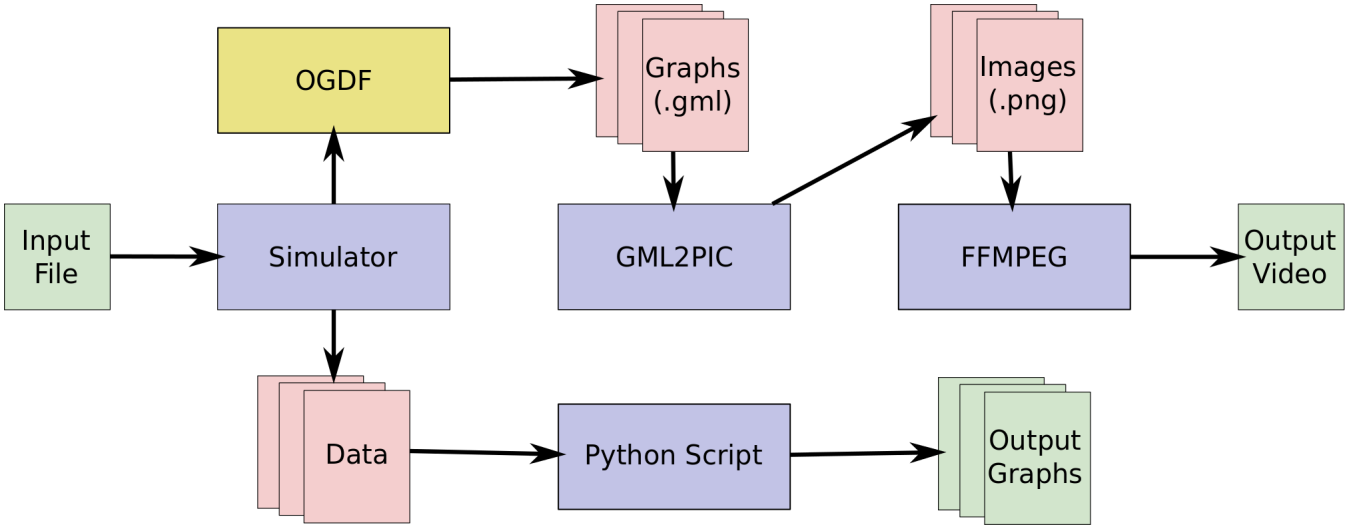

Figure 1: The workflow involved in generating the animation and graphs for our simulator. Green denotes input/output files. Red denotes intermediate files. Blue denotes programs that must be called. Yellow denotes major code libraries used.

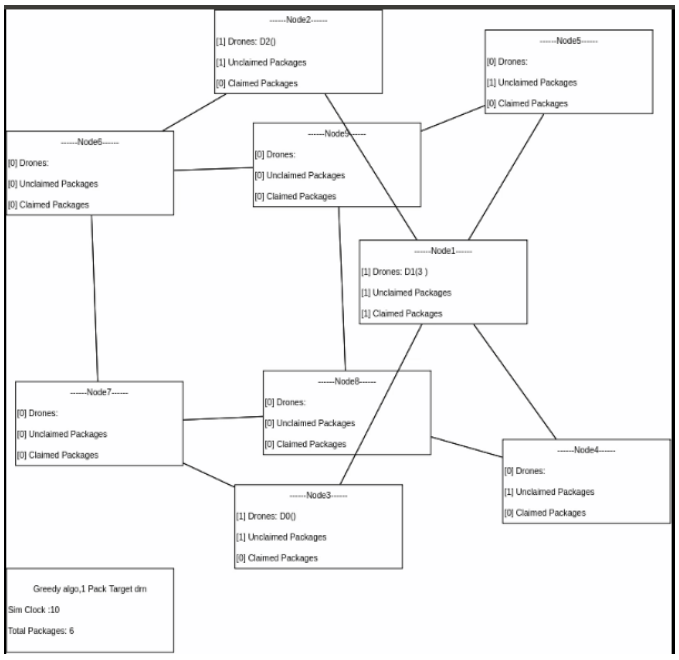

Figure 2: A screenshot of the final animation. While we used OGDF to generate our graphs, the graph algorithms did not always optomize for the most attractive layout.

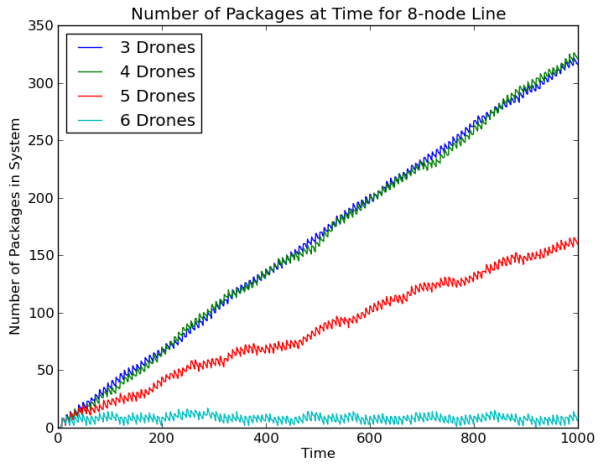

Figure 3: This graph shows the number of packages in our system at a given time. For this graph, we use a graph that consists of 8 nodes in a line with each node generating a package once every 8 time steps.

some drone needs to cover for that package is equal to the number of steps that drone needs to take to get to that package, plus the distance from the source to the destination node of that package.

Thus, since the total average distance that the packages require covering linearly increases (when the number of drones is below some threashold amount), we get the graphs that we see.

6. CONCLUSION

With this project, we explored some basic behaviors of a greedy routing algorithm on a variety of network graphs. To do so, we built an easily extendable, modular simulator capable of taking a variety of different options in an input file.

With the simulator, we showed that a simple, greedy algorithm works as expected. Additionally, we have also investigated a variety of different techniques relating to the vehicle routing problem.

7. FUTURE WORK

While we have done some basic, preliminary work with the given model, there is a large set of interesting and varied questions that one could ask with minimal changes to the model itself.

The best policies thus far seem to still be reliant on solving the Traveling Salesman Problem on some subset of the nodes in our graph. Would it be possible to come up with a $h(n)$-approximation algorithm (where $h(n)$ is the ratio between the cost of the approximate algorithm and the cost in the ideal) based on some other method?

In our greedy algorithm, we do not penalize nodes for moving freely between nodes. Thus, it would be possible to in-

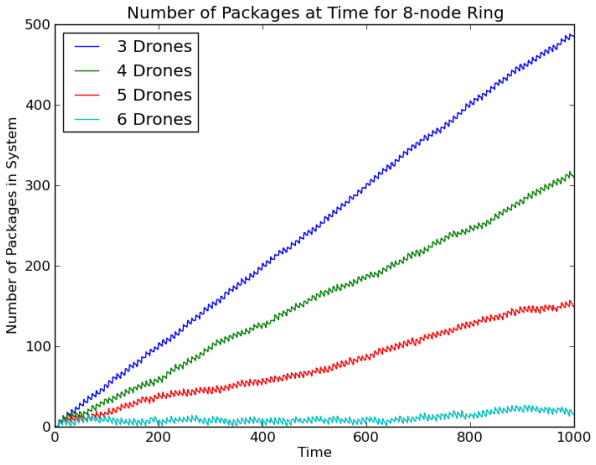

Figure 4: This graph shows the number of packages in our system at a given time. For this graph, we use a graph that consists of 8 nodes in a ring with each node generating a package once every 8 time steps.

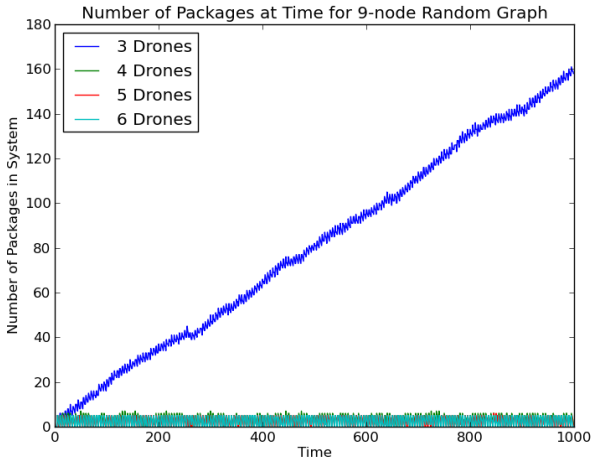

Figure 5: This graph shows the number of packages in our system at a given time. For this graph, we use a graph that consists of 9 nodes in a random configuration with 5 of the nodes generating a package once every 5 time steps.

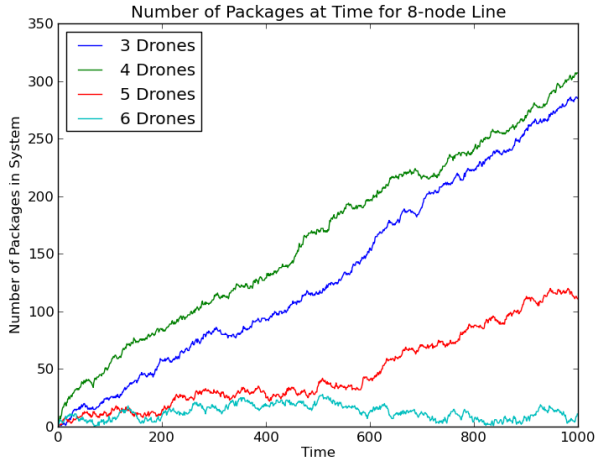

Figure 6: This graph shows the number of packages in our system at a given time. For this graph, we use a graph that consists of 8 nodes in a line with each node generating a package randomly with probability 1/8.

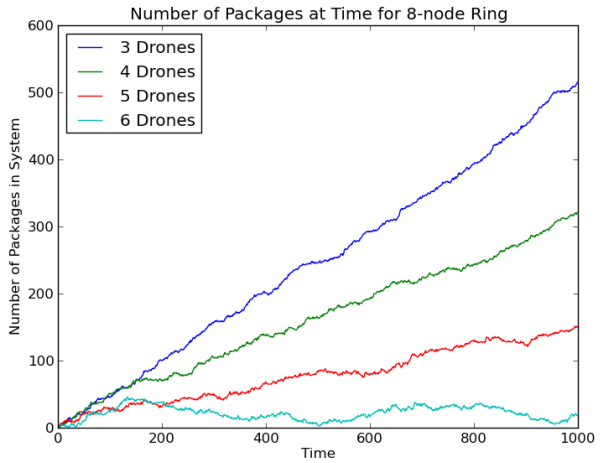

Figure 7: This graph shows the number of packages in our system at a given time. For this graph, we use a graph that consists of 8 nodes in a ring with each node generating a package randomly with probability 1/8.

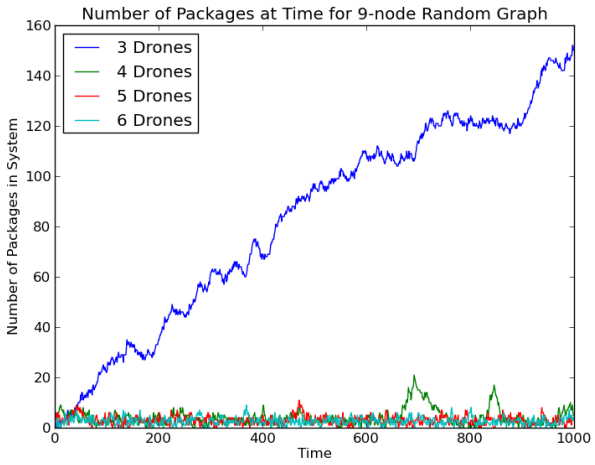

Figure 8: This graph shows the number of packages in our system at a given time. For this graph, we use a graph that consists of 9 nodes in a random configuration with 5 nodes each generating a package randomly with probability 1/5.

clude "send drones to node n_x while idle". How much would this increase or decrease the performance of the drones? What about if we included a cost function?

Our UAV package delivery system assumes knowledge of a full-snapshot of the system. What happens if drones are limited to only knowing the location of a subset of the packages and only a subset of the other drones' locations?

8. ACKNOWLEDGEMENTS

The authors would like to thank Qiuyu Peng and Steven Low for guidance and input. They would also like to thank Adam Wierman for organizing the CS 145 course. All funding was provided by the Computer Science department of the California Institute of Technology.

9. REFERENCES

[1] G. Laporte, Y. Nobert, S. Taillefer, "Solving a Family of Multi-Depot Vehicle Routing and Location-Routing Problems". *Transportation Science*. Vol. 22, No. 3, August 1988.

[2] Simulated Annealing `http://en.wikipedia.org/wiki/Simulated_annealing`

[3] G. Proietti, P. Widmayer, "Partitioning the Nodes of a Graph to Minimize the Sum of Subgraph Radii". *Algorithms and Computation*, pp 578-587.

[4] M. Pavone, E. Frazzoli, F. Bullo. "Adaptive and Distributed Algorithms for Vehicle Routing in a Stochastic and Dynamic Environment" *Automatic Control, IEEE Transactions on, (June 2011)*

[5] Amazon Prime Air. `http://www.amazon.com/b?node=8037720011`

[6] Quadcopter Pole Maneouvers. `https://www.youtube.com/watch?v=pp89tTDxXuI`

[7] Quadcopter Navigation. `https://www.youtube.com/watch?v=eznMokFQmpc`

[8] M.R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, E. Talbi. "Multi-environmental Cooperative Parallel Metaheuristics for Solving Dynamic Optimization Problems" *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on,* 16 to 20 May 2011

[9] V. Pillaca, M. Gendreauc, C. GuÃľreta, A. L. Medaglia. "A Review of Dynamic Routing Problems" *European Journal of Operational Research* Volume 225, Issue 1, 16 February 2013, Pages 1 to 11

[10] ED Taillard, LM Gambardella, M Gendreau, Y Potvin. "Adaptive memory programming: A unified view of metaheuristics" *European Journal of Operational Research* Volume 135, Issue 1, 16 November 2001, Pages 1 to 16.

[11] R. Montemanni, L. M. Gambardella, A.E. Rizzoli, A.V. Donati. "Ant colony system for a dynamic vehicle routing problem" *Journal of Combinatorial Optimization* Volume 10, Issue 4, 2005, Pages 327-343.