Exploring - Distributed Intelligent Dispatch

- Immediate, Scheduled, and Forecasted Requests
- Uncertain driver acceptance and availability
- Passengers, Packages, and Shared Rides

Given the real-time nature and extremely large volumes that ride-hail/delivery networks are dealing with, dispatch algorithms have traditionally relied on some form of Greedy assignment. In the traditional taxi space this amounts to zone based routing or nearest-driver. Although I know that TNCs has devoted extensive resource to Forecasting / Mapping / ETA components, I believe most are still using a Greedy Assignment algorithm – although a few may be analyzing batches of up to 8 rides combinatorially. However, as these TNCs moves into additional verticals like shared rides, scheduled rides, packages delivery, less densely geographic regions, and eventually autonomous vehicle networks, the nature of the problem may change, and efficiencies might be gained by using a more advanced "combinatorially thinking" dispatch algorithms.

The fact that we need to optimize under uncertainty makes this problem many times more difficult. How do we optimize in real-time when simulation is required to evaluate a solution? A key piece of Verint IP is their Skill-based patent (see references), which describes how to schedule when simulation is required. The Intelligent Dispatch problem is in real-time and much harder and won't lend itself to traditional Simulation Optimization techniques.

That computers are continuously getting faster, and with the parallelism that can be achieved with modern GPUs, I feel its becoming more and more feasible to combinatorially optimize in real-time at a large scale.

The following slides explore the various dimensions of this problem. What might distributed optimization look like? Region boundary and staleness issues. How do we deal with uncertainty? It proposes that we might solve for N complete problem instances (sampled from the forecast) and aggregate these in some way to make optimal decisions that will be good across the range of forecast data. It acknowledges the tradeoff between solving for many problem sample instances, vs. Clustering the forecasted data and solving against the cluster centers. It evaluates Combinatorial Optimization vs. Dispatch rules, and considers how Dispatch Rules might be learned. And finally, it considers what might be done between Optimization Runs, when waiting for the next run is not feasible.

A key idea is that we have between 1 to N layers, where each layer is the complete representation of the Intelligent Dispatch problem, from Now to 1-hour forward. For each representation instance, the "certain" data remains constant, but the "uncertain" (forecasted) data is populated by sampling from the forecast. Possibly hundreds of solvers, are continuously optimizing regions on different instance layers. At the moment that assignments must be sent to passengers and drivers, an Aggregator analyzes to "best" decisions across all instance layers, and dispatches the one with the highest expected value.

Driver / Passenger matching may be required before an optimization run has been completed for that data, in these cases, there must be a way to leverage the solutions from the most recent Scheduling/Optimization runs.

A complete solution may include many or all these components. There may be some requests that don't need to be routed that quickly, for others we may not have more than 1-second to come up with an adequate decision. The Distributed Optimization component, may be used (ignoring staleness) during off-line learning of Dispatch Rules, and also, as a Optimization component that completes every minute or so.

About me

My LinkedIn page: https://www.linkedin.com/in/edward-hamilton-b674a76

- Lead Architect behind Optimization Engine for leading Workforce Management Solution for Call Centers, Back-Offices, and Bank Branches.
  - scheduling over 1.5M agents per day
  - and $500M in annual revenue.
- Founded TripThru (a back-end interoperability hub for ride-hail networks) -- Failed.
  - We initially signed up some large tier-2 players, but the space ended up being driven by Uber growth and mergers and it became apparent that our model wouldn't work.
- Built conference scheduling system for Intel's world wide sales conferences.
  - Increase attendee satisfaction form 70% to 95% (measured by them getting requested courses)
- Developed custom language (IDE) and test environment for chip burning wave patterns and testing.
  - Sold rights to it for small potatoes (it paid my way through school). It did some cool in-memory / on-demand compilation stuff that made it faster than competitors.

Over 15 years experience developing forecasting, routing, scheduling, and planning solutions for service industry.

Dimensions of the problem

Distributed Optimization. Slides 4-11

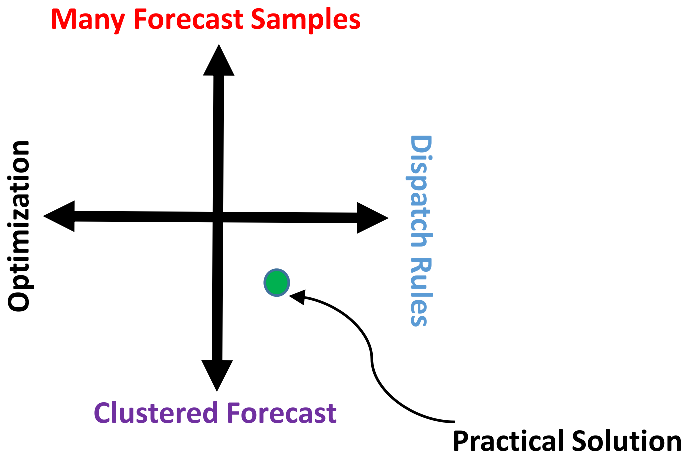
Solving for many Problem Sample Instances and Aggregating Solutions (Slides 12-19)
vs.
Merging/Clustering Forecast Samples and Solving Against Cluster Centers (Slide 20)

What to do between Optimization Runs (Slide 21)

Combinatorial Optimization
vs.
Dispatch Rules (Slides 22-24)

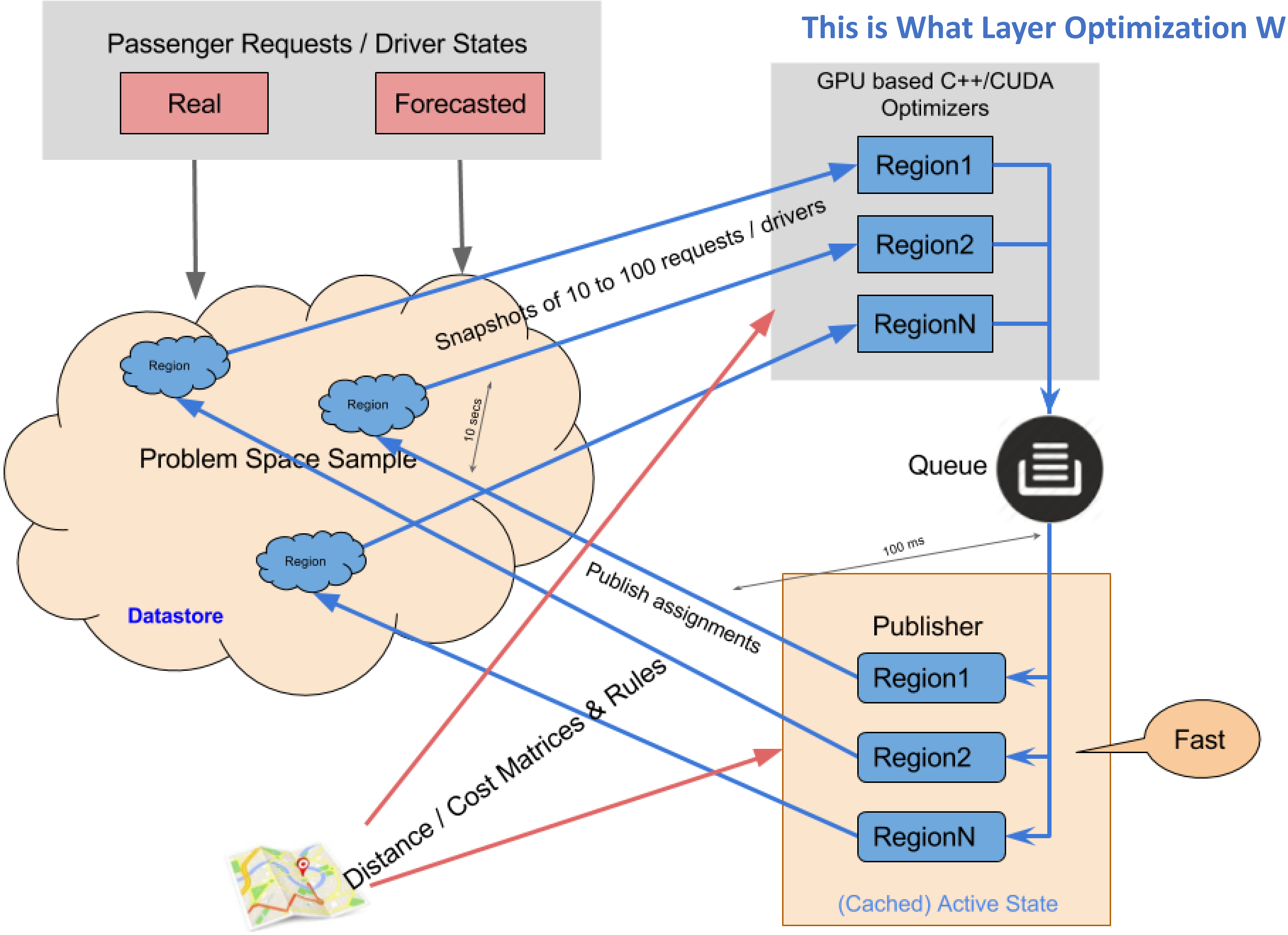
Roadmap (Slide 25)

References (Slide 26)



Distributed Optimization

This is What Layer Optimization Will Look Like



Edward Hamilton -- https://www.linkedin.com/in/edward-hamilton-b674a76

The following four slides attempts to visualize how regions are selected and to acknowledge that there are edge conditions that need to be considered. It proposes that the area input to the region optimizers will be larger than the publish area, in order to smooth out the edges.

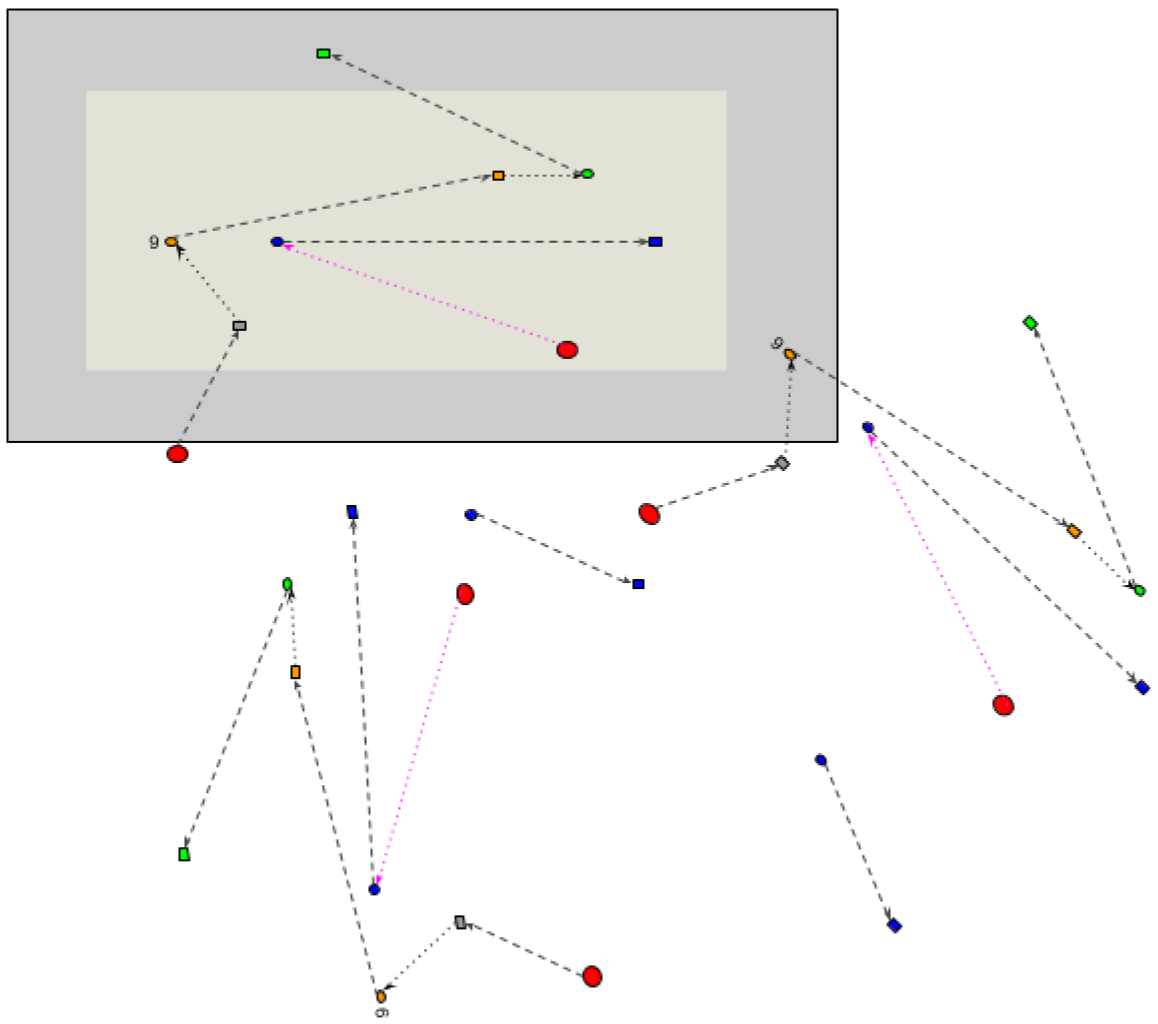
(Simple) region selection algorithm: We want regions that generally overlap many pickup / drop-off pairs to maximize intra-region routes. We add noise because sometimes it will be optimal for routes to cross these pickup/drop-off pair clusters. This is for discussion sake. Once we look at the data, we can improve upon this.

1. Randomly select a coordinate C on the map.
2. Randomly select a radius R between X and Y
3. Select noise level
4. Move circle of radius R around the map, maximizing contained pickup / drop-off pairs -- constrained by C must be in the circle

Region Boundaries

Snapshot 1

Distributed Optimization



▨ Locked region

□ Unlocked region

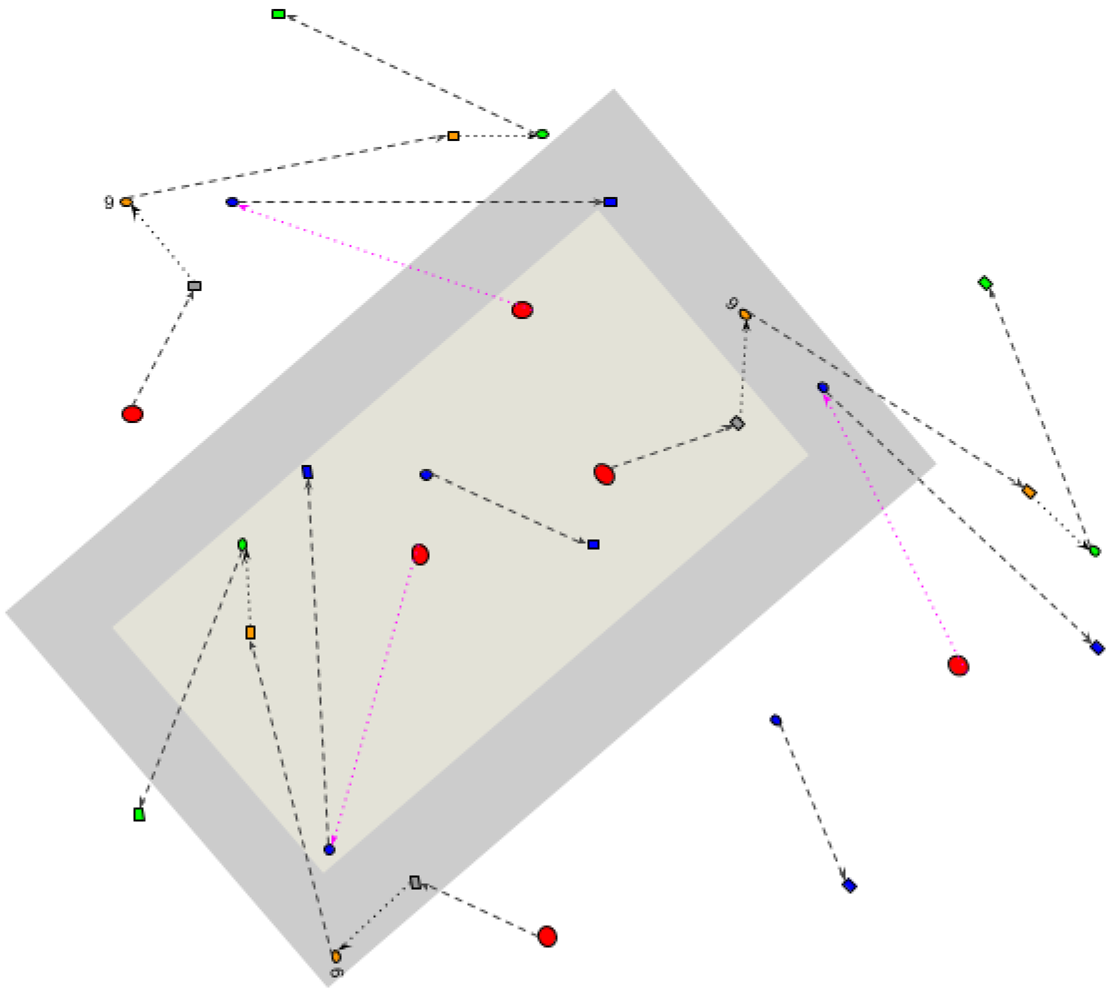
Edge Types

- Driver In - Request Out
- Request In - Driver Out
- Request & Driver In

Distributed Optimization

Region Boundaries

Snapshot 2



▨ Locked region

□ Unlocked region

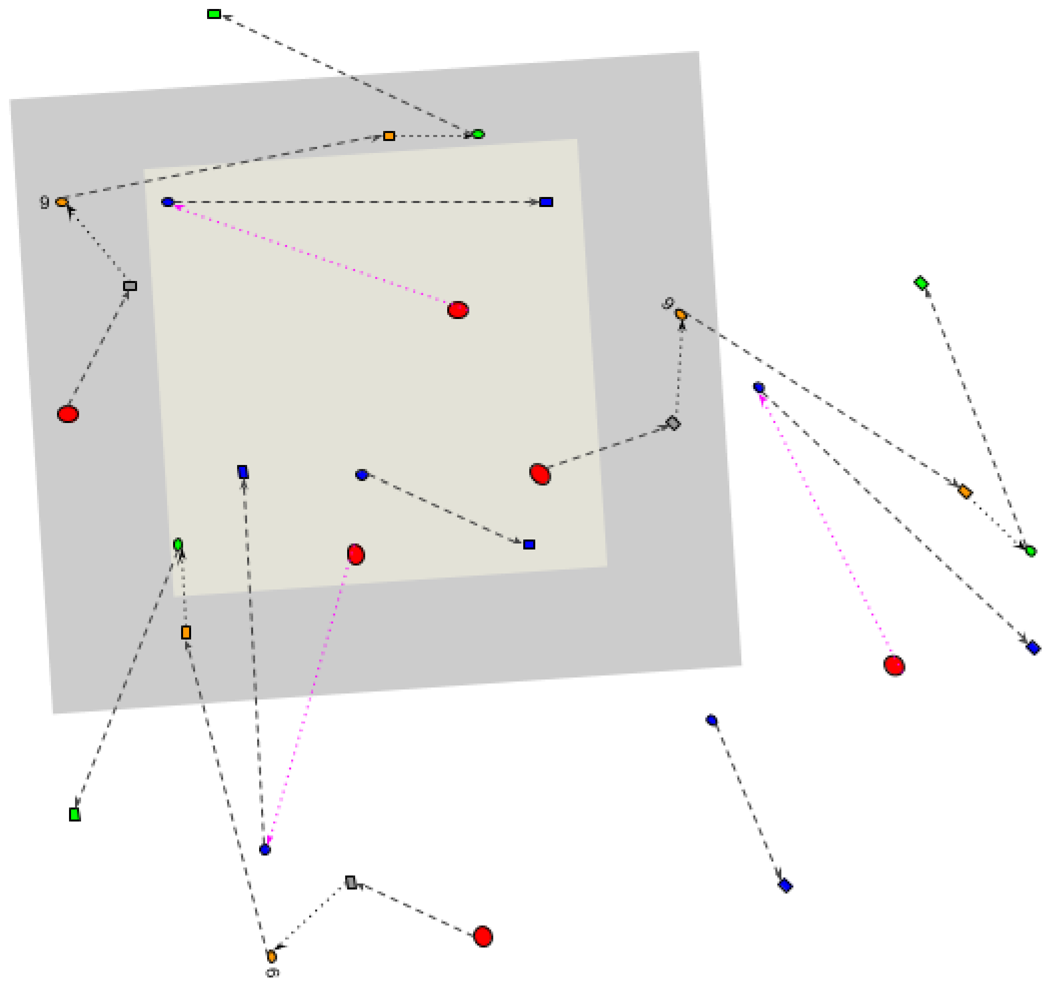
Edge Types

- Driver In - Request Out
- Request In - Driver Out
- Request & Driver In

Region Boundaries

Snapshot 3

Distributed Optimization



▨ Locked region

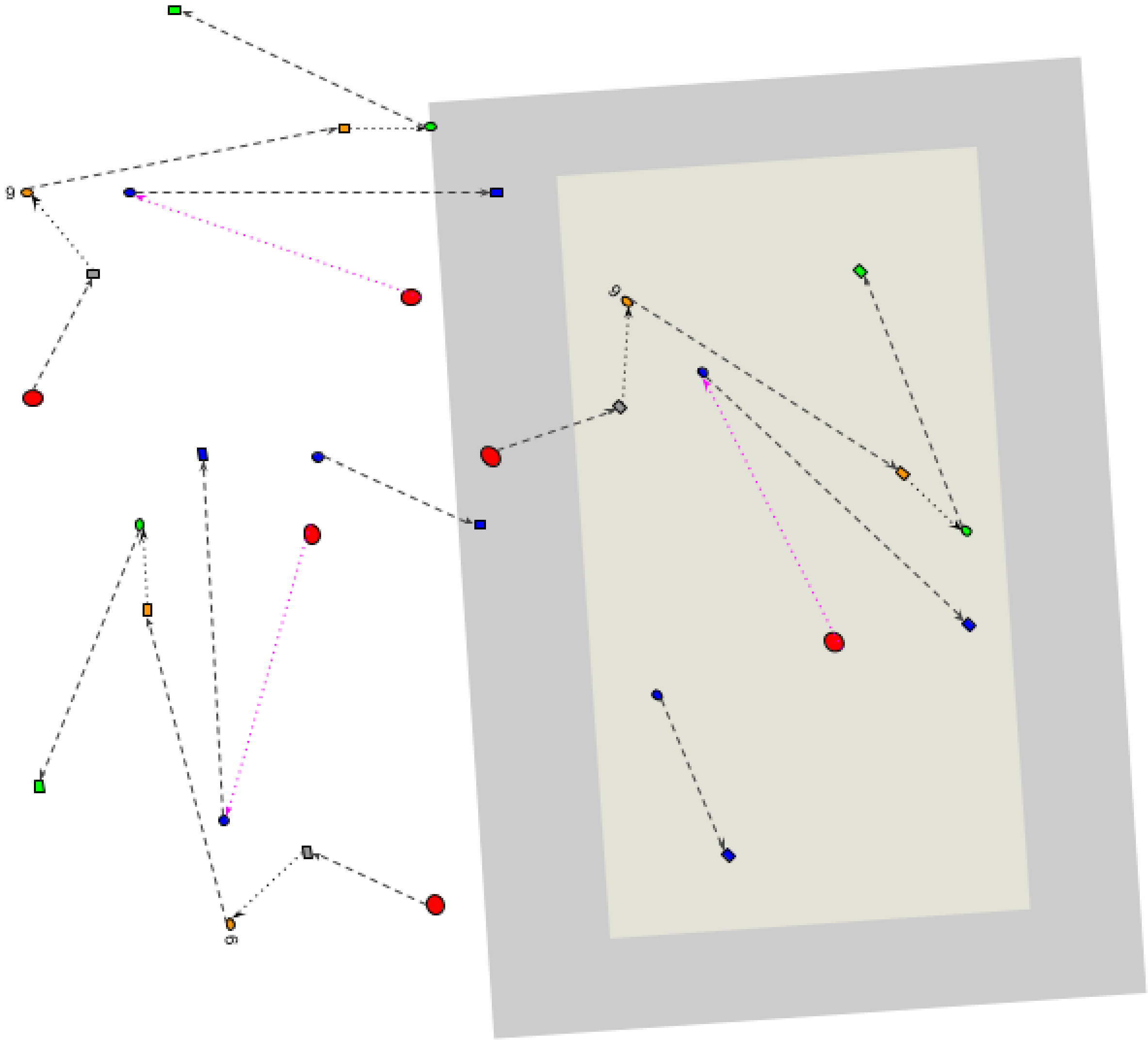
□ Unlocked region

Edge Types

- Driver In - Request Out
- Request In - Driver Out
- Request & Driver In

Region Boundaries

Snapshot 4



▨ Locked region

□ Unlocked region

Edge Types

- Driver In - Request Out
- Request In - Driver Out
- Request & Driver In

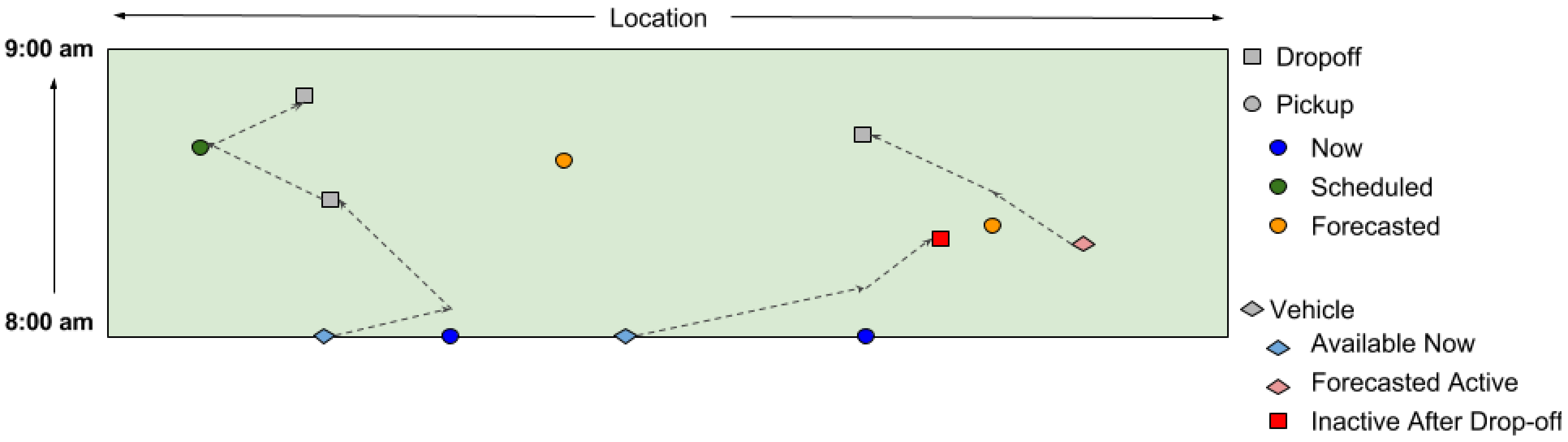
Distributed Optimization

Description: This is What Layer Optimization Will Look Like

- **Problem space sample / layer**: This data-store contains both real-time and forecasted passenger & driver data. For now let's assume this is for a single city and a 1-hour time period. This space is being continuously optimized. Although not shown here, an Aggregator will analyze decisions across all problem layers / instances and select the decision with highest expected value. Agents act upon the best assignments available at that moment.
- **Request types** are: **Now**, **Scheduled**, and **Forecasted**. They could be passenger pickup, package delivery, shared rides, scheduled ride -- whatever. Even multi-modal requests.
- **Map/ETA** module is considered given. It knows how to get a driver from a to b, costs, ETAs, rules, etc. Uber has their down Map/ETA module, but for many other ride-hail/delivery networks, this will be Google Maps.
- **Assignment Optimization Worker Servers**: are continuously taking snapshots of spatial / temporal regions and optimizing the assignment of requests to driver. These workers will likely be written in C++/CUDA and heavily parallelized using GPUs.
- **Publisher**: Since the optimization of each region could take between 10 seconds to minutes (configurable), the snapshot may become stale by the time the worker is complete and ready to publish. Likewise, there may be many workers optimizing overlapping regions. It is the job of the publisher to make sure that only legal and best solutions are published. Perhaps there is something like a "merge" operator that can optimally combine multiple overlapping solutions.
- **Aggregator**: Not shown until slide 19, but there is a process that aggregates the layer sample solutions to produce best 1st step decisions.
- **Configuration:** Most obvious configuration parameters are:
  - Region size: We will optimize many overlapping regions, and larger regions size -- deeper search but slower.
  - Region Optimization Time Limit: the Time Limit must be small enough to minimize the staleness issue. This architecture should lend itself to adding many multi-core multi-GPU "worker" servers.
  - Weight Forecasted Data
  - Number of Problem Space Layers/Samples

2D Region 1 Hour Instance Slice

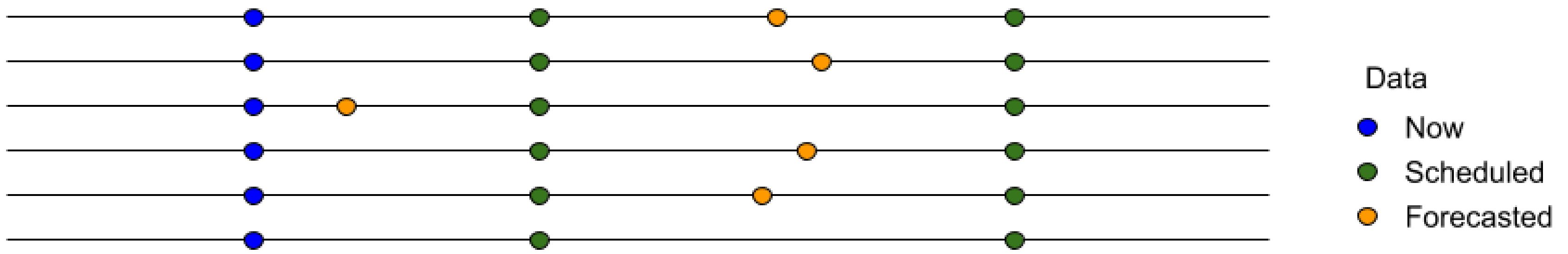
This slide attempts to visualize the optimization look-ahead window.   It corresponds to a region in previous slide/figure



- This is a 2 dimensional slice (for visualization) of a region 1 hour schedule ahead window
- There are Immediate, Scheduled, and Forecasted requests coming in.
- In this example, 8am is “now”, where we have a perfectly accurate picture of the world
- As we look ahead in time, our picture of the world becomes partially forecasted -- uncertain.
  - New unscheduled requests come in -- which we may be able to forecast to some degree
  - Drivers become available and others become unavailable.
  - However, we may have some known data as well. Scheduled requests, drivers who will be at a specific drop-off location
- Ultimately we’re optimizing for decisions that need to be made now.   We optimize 1 hour into the future so our decisions will take the future into account.

Solving multiple (forecasted) problem sample instances and merging solutions

1D Region 1 Hour Instance Input Data Slices 8am - 9am

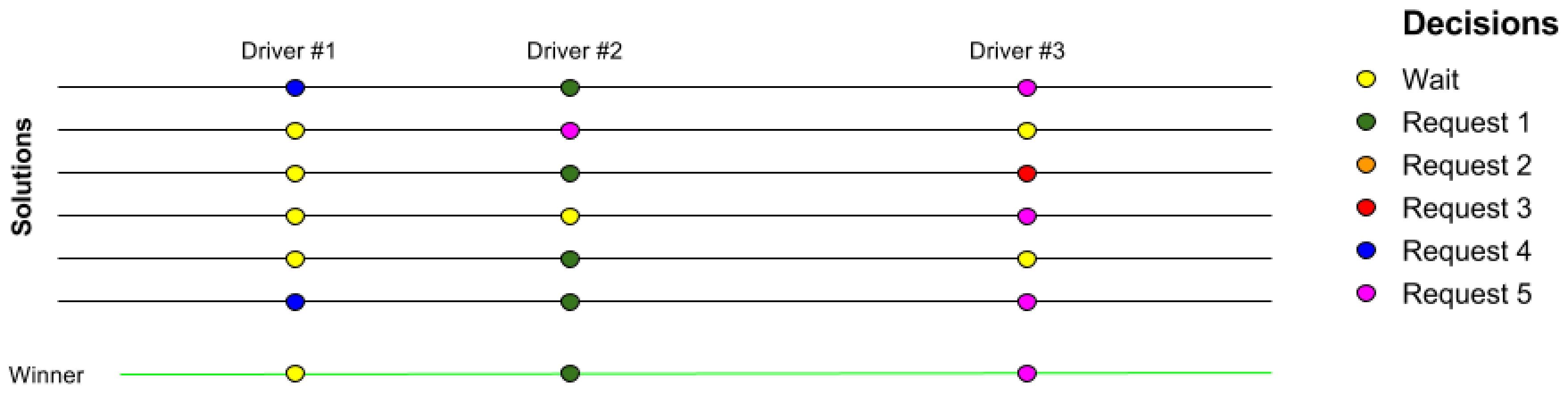


- So we have 1 hour region containing both certain data and uncertain data.
- We can optimize these region instances separately. See instance optimization slide.
- The solutions will deviate between instances according to how much the sampled (from forecast) data deviates between instances, and how much weight we give to the forecasted data vs. the certain data.
- We can optimize these instances separately and we end up with N separate solutions.
- However, we only use the immediate (now) part of the generated solutions.
    - An assigned pickup request
    - Or Wait
- The route chains are built out 1 hour into the future, to help us make decision for “now”.   But by the time drivers need to act on the future decision, they may have been replaced by better decisions / routes.

In this slide, we’ve collapsed the look ahead time window so that we can show may layers at once.   The point that I’m trying to get across is that the Immediate (Now) and Scheduled data remains the same across all layers, but the forecasted data is changing.

Solving multiple (forecasted) problem sample instances and merging solutions

1D Region Slice of Best Decisions for 8am



- We maintain N full route assignment problems and solutions, where each solution is built from actual data, but also forecasted data. Where for each problem instance the actual data is constant across all instances, but the forecasted data is sampled from the forecast.
- Then there's a process that analyzes the best decisions across all sample instance to produce a "best" decision to be acted on. A simple implementation would be to take the solution that is returned most frequently across solution instances or the one with the highest expected value.

In this slide, we're emphasizing the solutions rather than the input data. I want to show that we analyze the decisions across all layers and use the one that yields the highest expected value. We can imagine some sort of aggregator doing this job. I admit that there seems to be a bit of voodoo here. Although layers are globally optimized, the Aggregator is making local decisions.

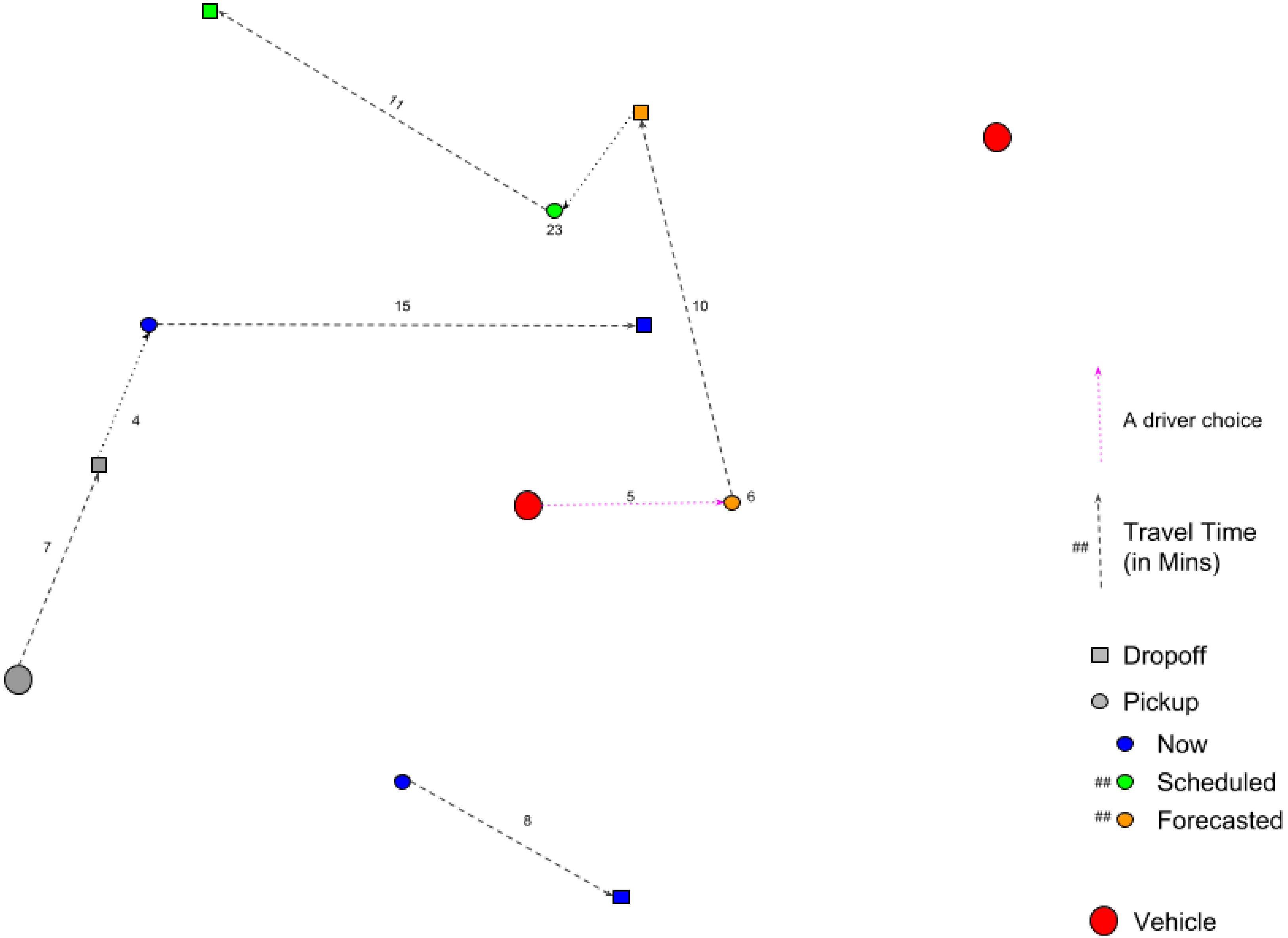
Solving multiple (forecasted) problem sample instances and merging solutions

The following four slides are another way to look at the layers in slide 12&13.   It’s a 3d rendition, and if I was making a video, I’d use something like them as a basis for an animation.   The main point is that there is a decision that occurs more often when considering all layers.   In this example, it’s the decision for driver to wait for a forecasted request instead of getting immediately assigned an existing request.

Region Data Instance (Real & Forecasted)

8:00 am **Sample 1**

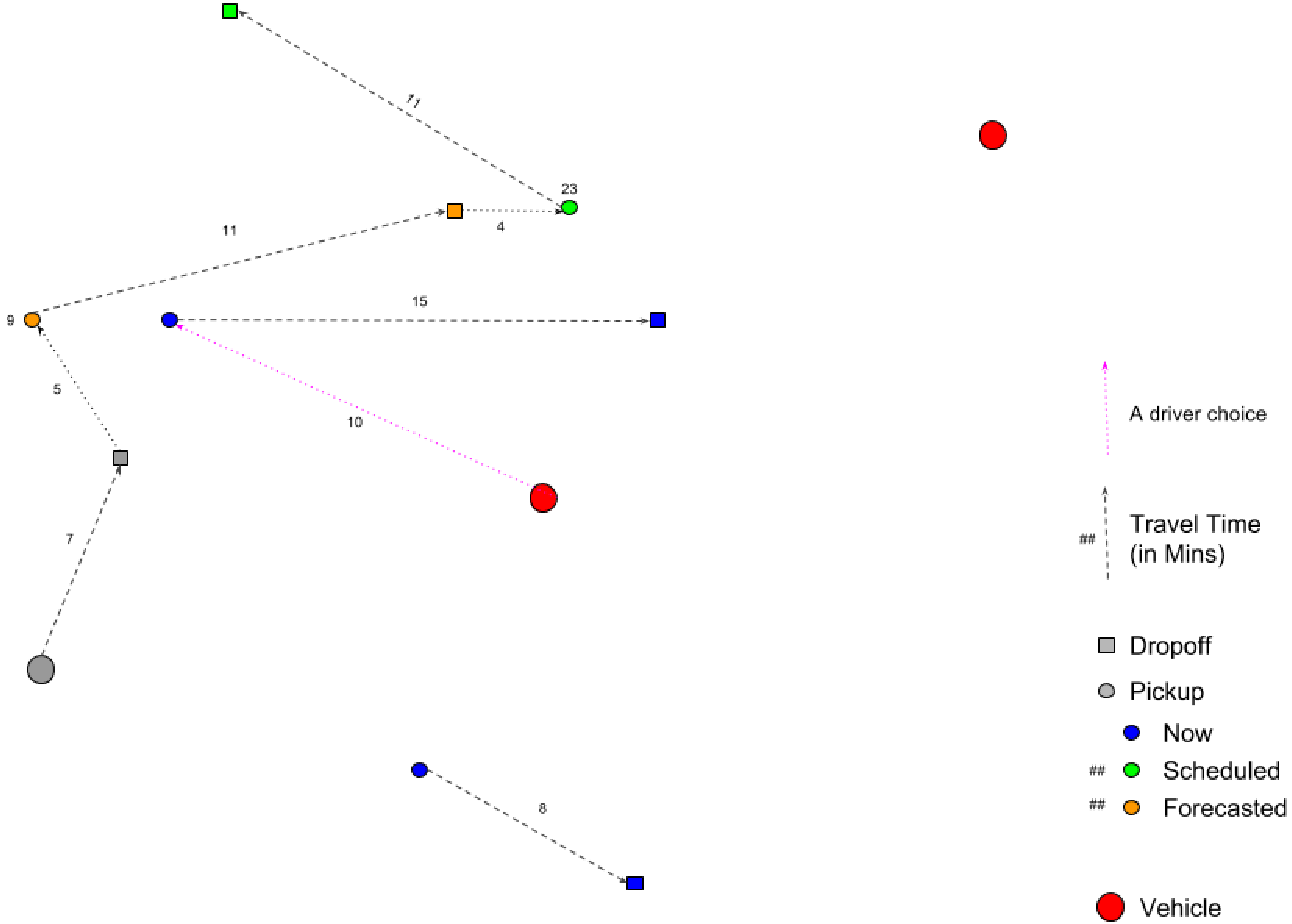
Solving multiple (forecasted) problem sample instances and merging solutions



Region Data Instance (Real & Forecasted)

8:00 am **Sample 2**

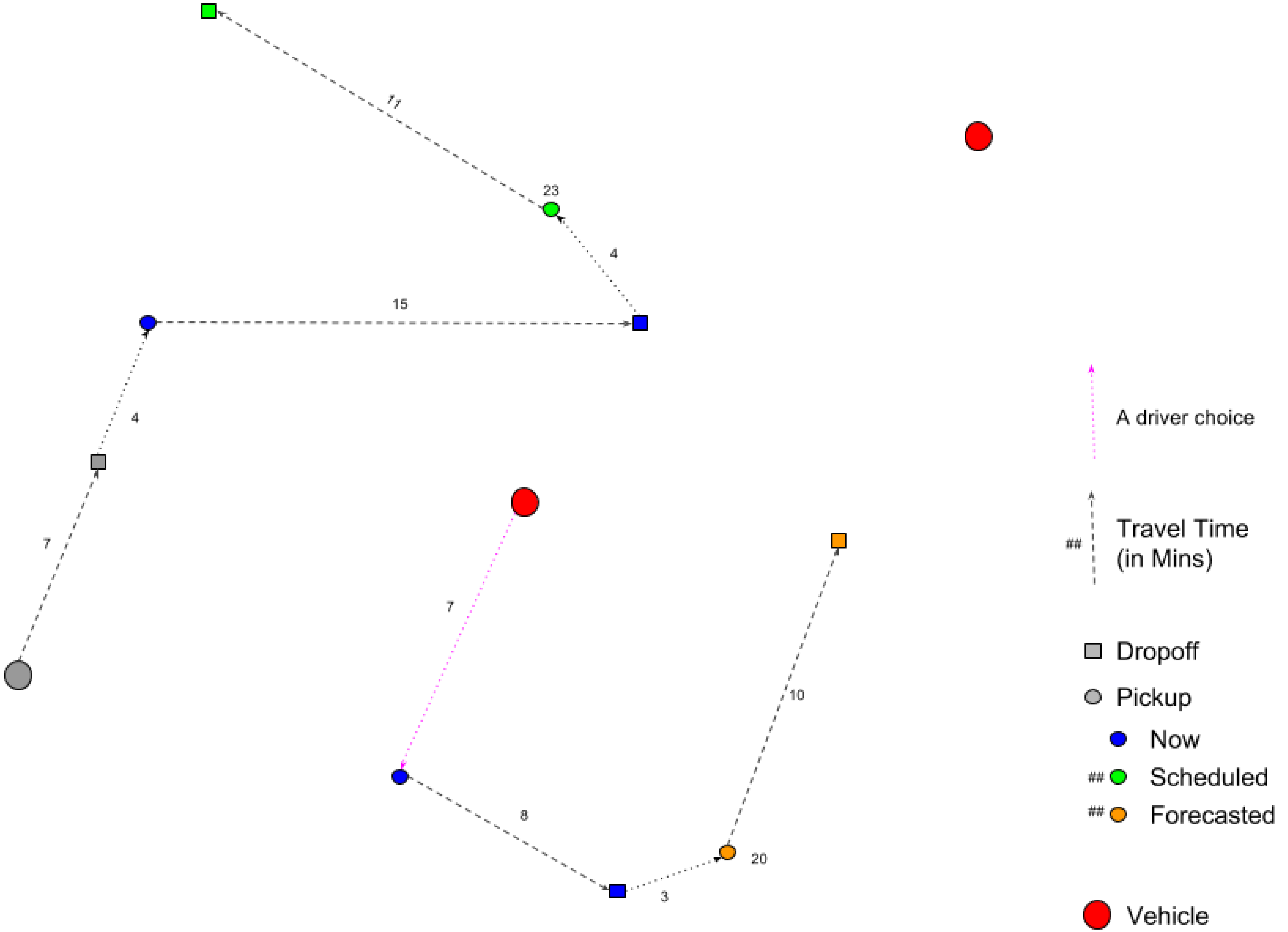
Solving multiple (forecasted) problem sample instances and merging solutions



Region Data Instance (Real & Forecasted)

8:00 am **Sample 3**

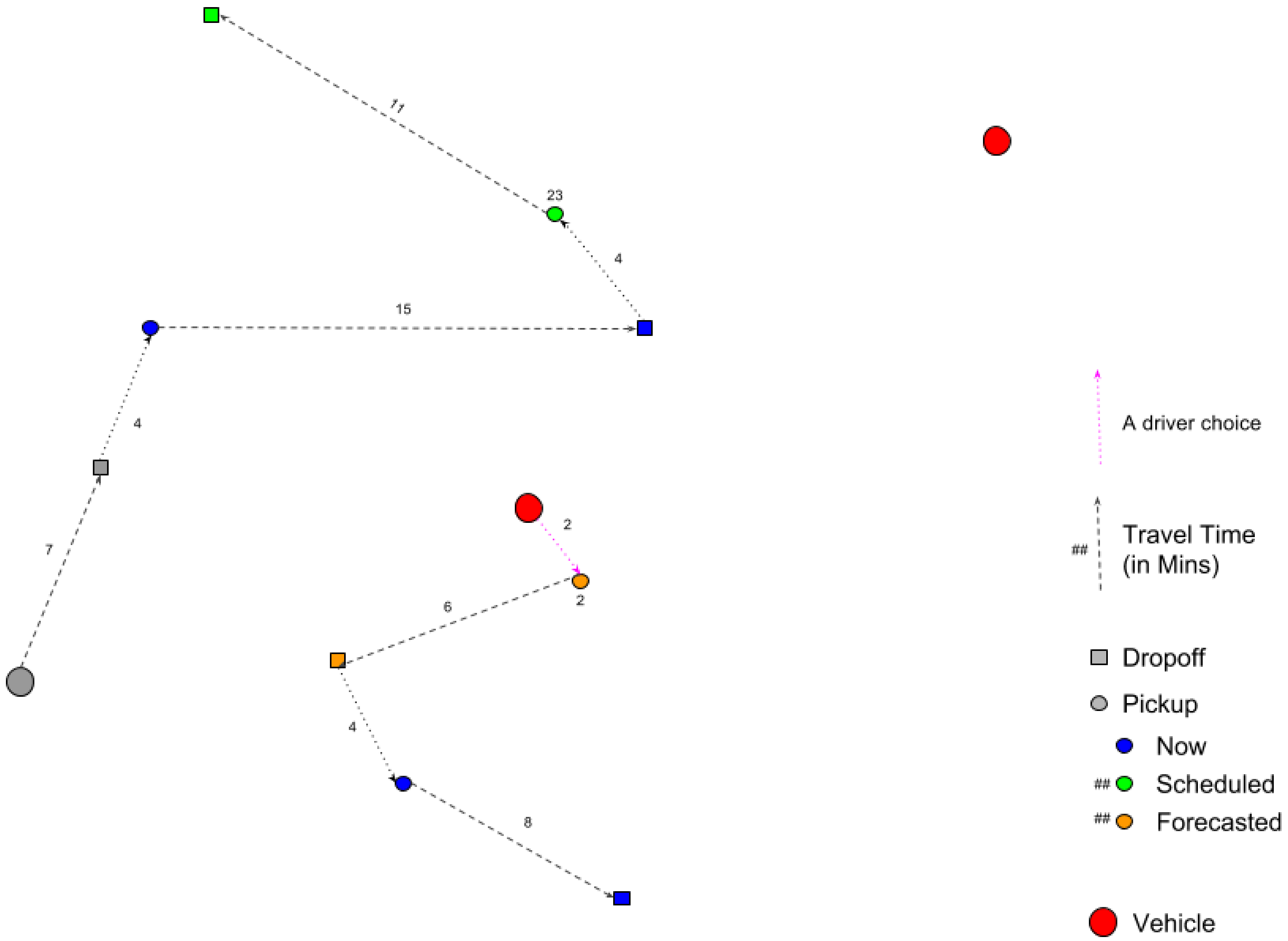
Solving multiple (forecasted) problem sample instances and merging solutions



Solving multiple (forecasted) problem sample instances and merging solutions

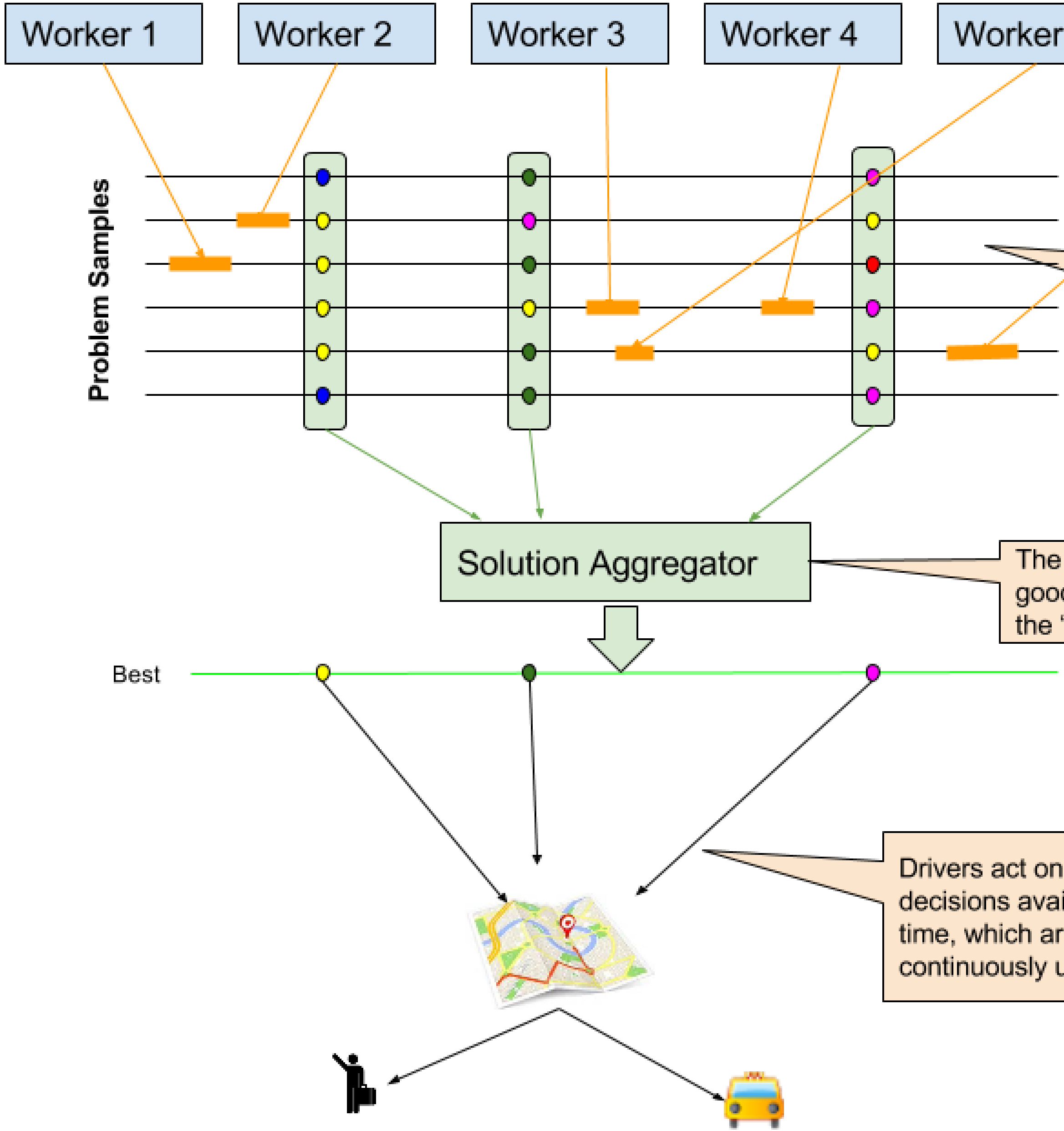
Region Data Instance (Real & Forecasted)

8:00 am **Sample 4**



1d Region Instances Being Optimized by many servers

Solving multiple (forecasted) problem sample instances and merging solutions



Each slice represents the complete problem data and solution, where the immediate and scheduled data is constant across all instances but the forecasted data is sampled from the forecast

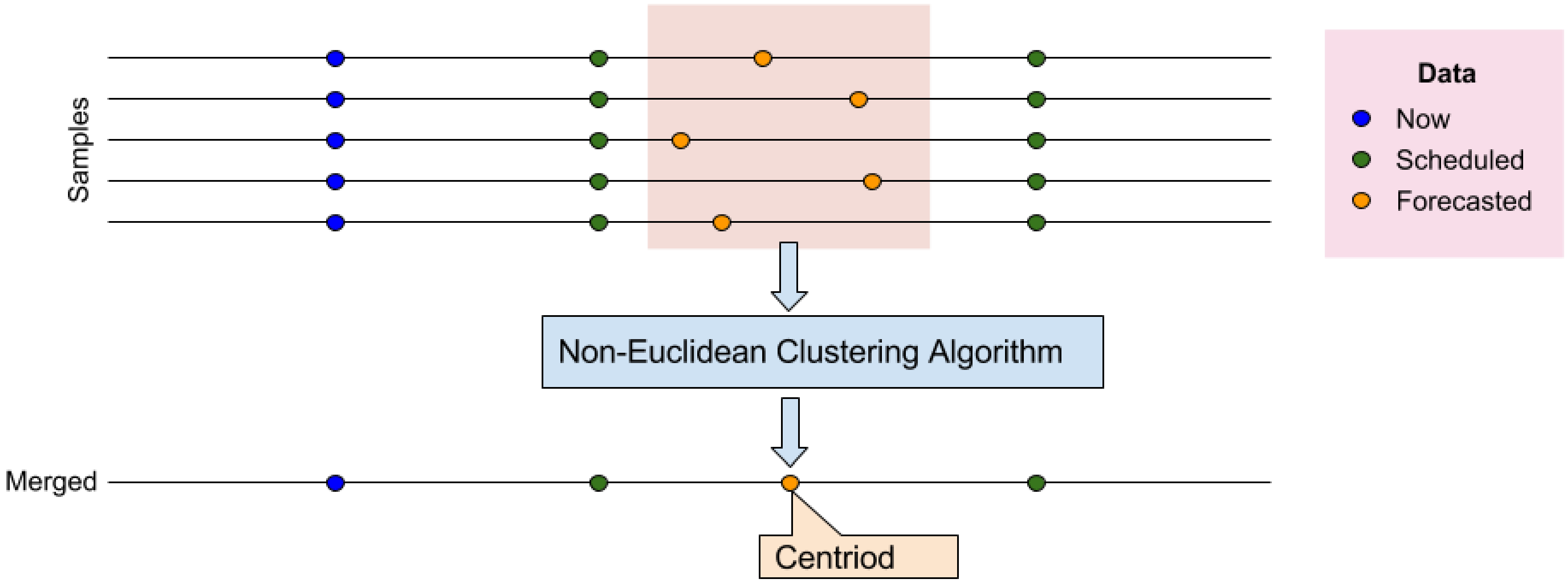
The “Solution Aggregator picks decisions that will be good across the possible forecasted data, as well as the “certain” data (Now & Scheduled requests)

Drivers act on the best decisions available at the time, which are being continuously updated.

In this slide, I’m trying to show a Birdseye view. Each layer/slice is being continuously optimized by system in Slide 2. When an assignment must be made, the aggregator looks at the range of decisions across all layers (for that assignment) and picks the one with best expected value.

Spatially merging the forecast (input) data to reduce number of sample instances

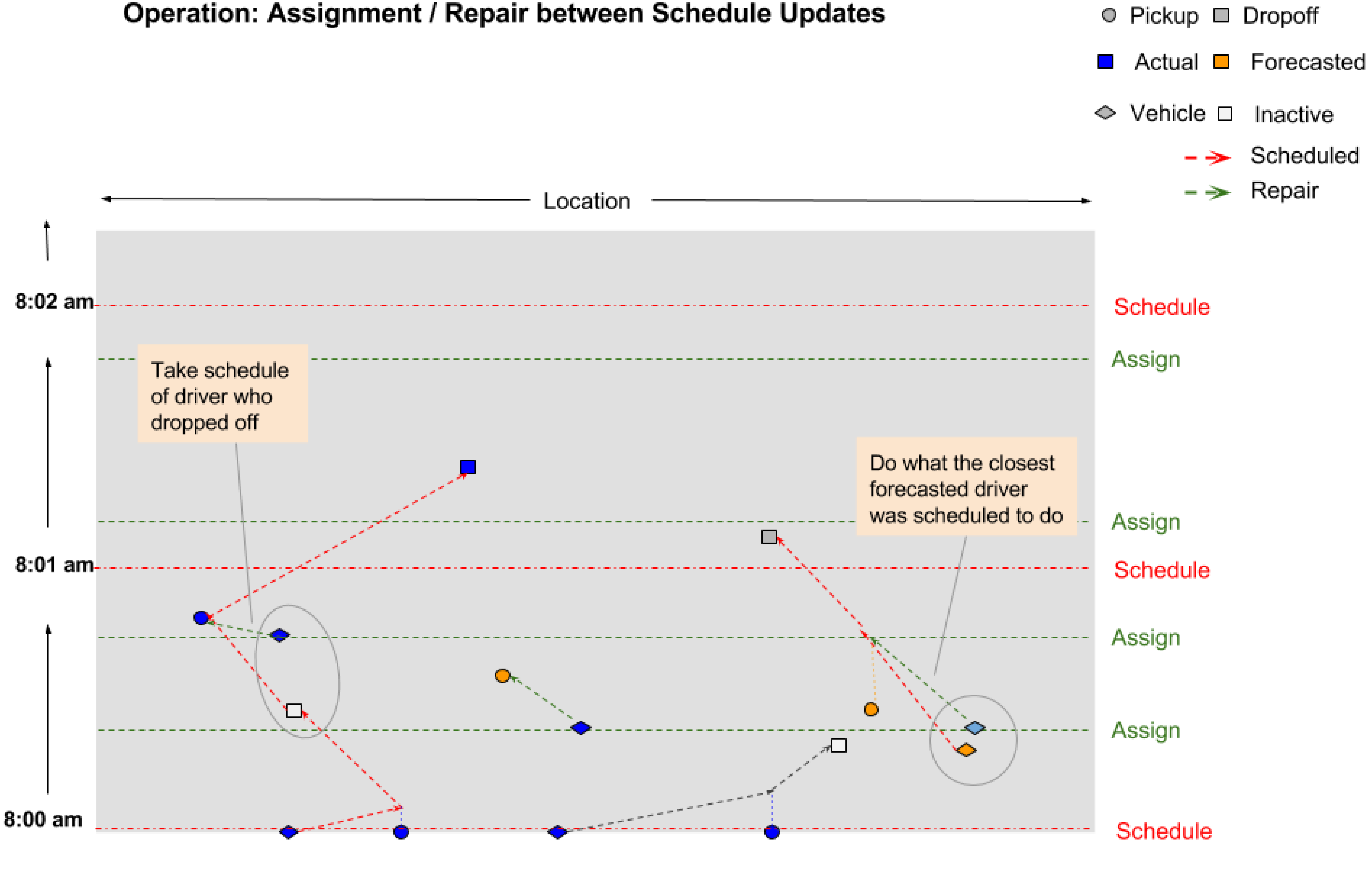
Reducing # of Problem Instance Samples / Clustering Forecast Data



We'll have to balance number problem instance samples. At the extreme, we could merge all problem instance samples into one instance. In this case, we're optimizing forecasted data centriod as if it was certain data.

- Need to think about forecasted requests vs. driver availability data.
- Do we have to treat it differently?

Operation: Assignment / Repair between Schedule Updates



Take schedule of driver who dropped off

Do what the closest forecasted driver was scheduled to do

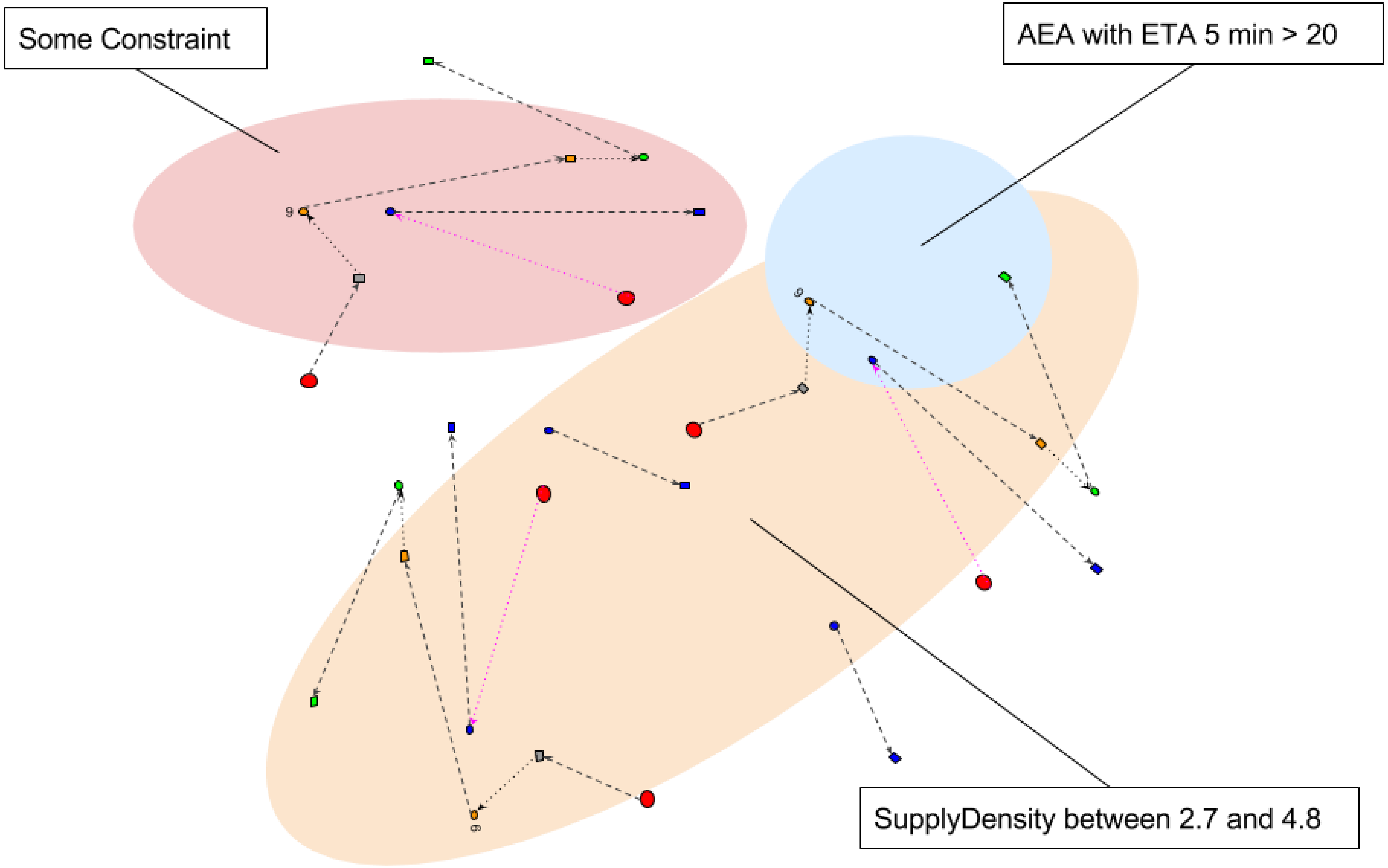
The idea is that we adhere to the schedule as close as possible, but allowing for real-time flexibility

Spatially merging the forecast (input) data to reduce number of sample instances

Some Definitions

- **Skills matching:** Some locations would be better served by different types of driver/couriers. For example a bike courier when parking is difficult. Average Handling Time (AHT) may be a way to model this. So if pickup/drop-off AHT is factored into the total cost, we should be OK. Perhaps this would result in something like Skill Per Location or LocationType (should be a way to learn types). I guess, if we have a courier type, then it will be easy to learn the Default AHT for this type. Then driver/couriers might have a Proficiency associated with different types of locations / jobs. Proficiency = AHT(Courier, LocationType) / AHT(CourierType, LocationType). Travel time will be adjusted by Proficiency. Anyway, in the Call Center world, ACDs collect this sort of data for us, which we can then use in optimization.
- **AEA (Average Effective Available Drivers/Couriers)**: Driver / Courier of average proficiency for that skill type and / or location.
- **SupplyDensity(region)** =$\sum_{Agents}\frac{AEA(agent)}{ETA(region)}$
- **Possible Constraints**:
    i. At least x AEA within Y minutes of some geographic region.
    ii. SupplyDensity(regionA, timerange) between X and Y.

Constraints Based Routing: 8:00 - 8:01am



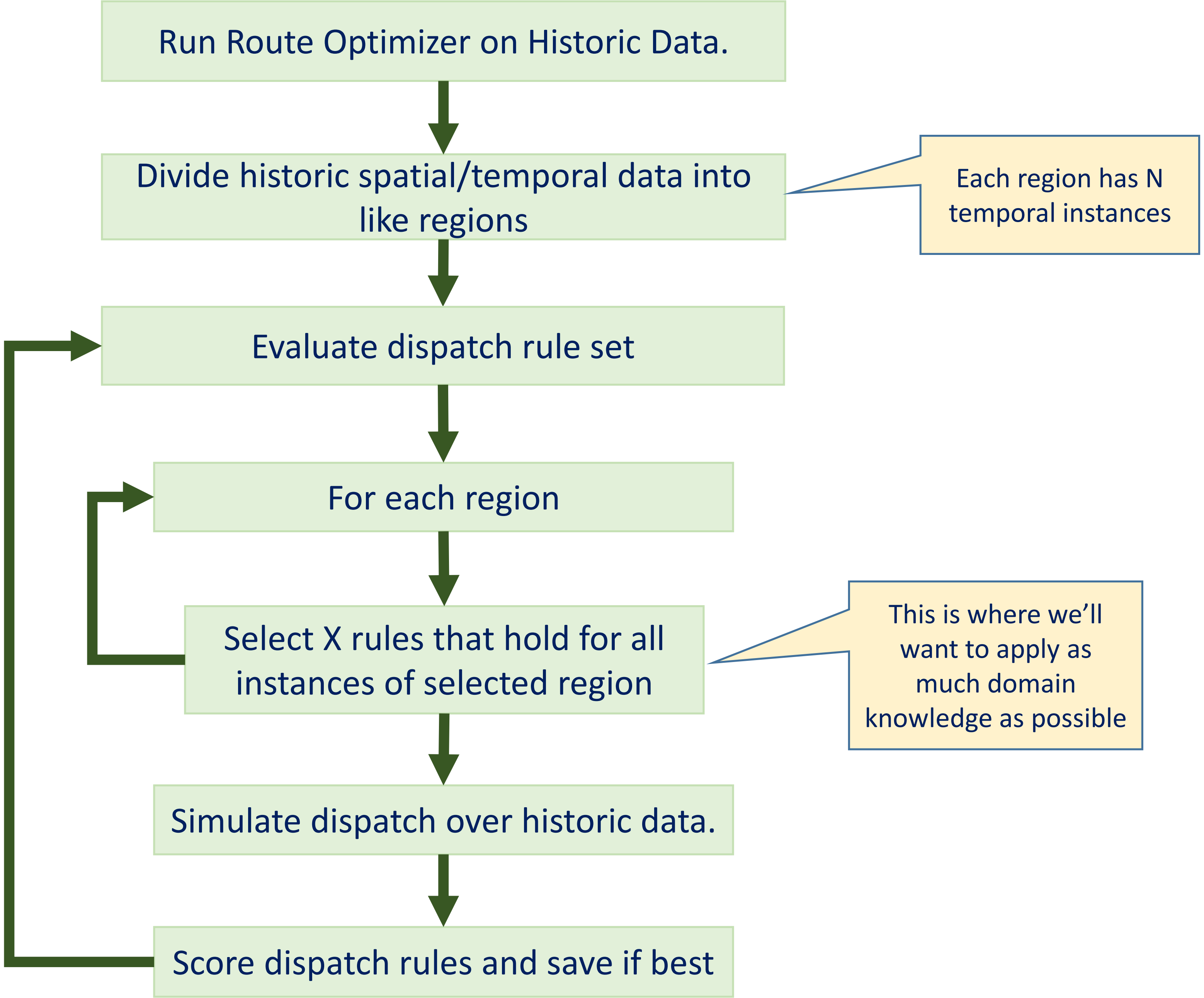
Some Constraint

AEA with ETA 5 min > 20

SupplyDensity between 2.7 and 4.8

Learning Dispatch Rules by analyzing optimal solutions

Learning Spatial / Temporal Dispatch Rules



Run Route Optimizer on Historic Data.

Divide historic spatial/temporal data into like regions

Each region has N temporal instances

Evaluate dispatch rule set

For each region

Select X rules that hold for all instances of selected region

This is where we'll want to apply as much domain knowledge as possible

Simulate dispatch over historic data.

Score dispatch rules and save if best

Roadmap

Before embarking on a large project like this, we'll need to do some POC with the actual data. Initially, we won't worry about performance, so no distributed systems, etc.. We will determine $ saving gained through advanced optimization, starting with best case (ignoring performance and uncertainty) and gradually adding constraints and considerations, measuring $s saved at each stage, until we reach a fully realistic solution.

1. **Ability to forecast**: Measure how well we can forecast. I need to speak to your Data Scientist about this. Simplistically, maybe we want to measure something like MAPE see reference (5) with a spatial MAPE formulation. Ultimately, we need to see how well we can optimize against the forecasted data, but this will give us some qualitative sense of the data. We don't necessarily have to do this first. We could also, validate optimization against a predicted forecast error (possibly).
2. **Best case value of advanced optimization**: First, we can test out our VRP(DARP) optimizer against the actual data, pretending there's no uncertainty. There are plenty of algorithms out there, and we don't have to worry about performance, overlapping regions, and all the messiness of a "real" solution. If we could perfectly predict the world and ignore performance, how much $s can we save, through advanced optimization.
   a. **Staleness issue**: At this point, we should write a crude Publisher, ignoring the region boundary issues. We can statistically plot the relationship between region optimization runtime vs. "probability rejection due to staleness". We can use historic data to measure this.
   b. **Given runtime constraints:** Now we may want to consider performance given the runtime vs. staleness plot. Given reasonable runtime limits, ignoring uncertainty, how much $s can we save through advanced optimization.
   c. **Between Optimization**: See slide 20. We can experiment with assignment / repair algorithms between simulations / optimizations, and reconsider our best case $s saved.
   d. **Region Boundary issues**: Enhance Publisher to handle the region boundary issues. How, much does this eat into our savings?
3. **Consider uncertainty**: Next, we can start testing with uncertain data.
   a. If we have a Forecaster, we can generate N problem instance samples, for some historic period, if we don't, then we can just take the historic periods and add random error (move existing data-points in space and time, and remove and add other data-points).
   b. When we get to the scheduling, we can play with a couple of dimensions – "Solving the N problem instances separately and aggregating the results" .vs. "Merging the N problem instances into a single problem instance to be solved".
4. **Learning Dispatch Rules**: We can also start to analyze whether all the good solutions have some properties that can be used as constraints for the real-time router. The advantage of the dispatch rule approach, is that the learned rules can be applied on top of the existing dispatch system, and hence could get into production quicker.

References

1. Various:
   a. Intelligent Dispatch System for advanced Reservations: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.680.3245&rep=rep1&type=pdf
   b. Compares Greedy to other dispatch algorithms: https://blog.routific.com/taxi-dispatch-algorithms-why-route-optimization-reigns-261cc428699f#.cpsd9q4ix
   c. Combinatorial Optimization vs. Dispatch Rules: http://www.davidmontana.net/papers/mista05.pdf
   d. Learning Dispatch Rules by Evaluating Optimal Solutions: https://notendur.hi.is/hei2/presentations/lion5_linearJSP.pdf
   e. Distributed Optimization: http://www.ifp.illinois.edu/~angelia/dssm_lip_cdc09revised_submit.pdf
   f. Taxi Demand Forecasting: https://serv.cusp.nyu.edu/projects/demandprediction/2016taxidemand.pdf
2. How will we optimize the local region?
   a. https://en.wikipedia.org/wiki/Iterated_local_search
   b. http://cepac.cheme.cmu.edu/pasi2011/library/cerda/braysy-gendreau-vrp-review.pdf
   c. https://www.itu.dk/people/pagh/CAOS/DARP.pdf
   d. http://www.learningace.com/doc/6042728/75eb3c11ae4768baa41e55f31436d177/a-two-stage-heuristic-with-ejection-pools-and-generalized-ejection-chains-for-the-vehicle-routing-problem-with-time-windows
3. Simulation Optimization
   a. http://www.solver.com/simulation-optimization
   b. https://en.wikipedia.org/wiki/Simulation-based_optimization
4. VRP Optimization Exercises - Region optimization could be based on something like this (I'm working on a more general solver that would support the other problem variants). There are many good VRP algorithms out there (see #3d).
   a. Source code: https://github.com/edwardhamilton/Optimization-Exercises
   b. Animation: https://www.youtube.com/watch?v=tKCJ9V8tEd4
5. Verint's Skill-Based Scheduling Patents – a solution for optimizing when Simulation is required to evaluate a solution
   a. https://www.google.com/patents/EP1248448A2?cl=en
   b. https://www.google.ch/patents/US20020143597?utm_source=gb-gplus-sharePatent
   c. http://www.google.ch/patents/US20050125439