Optimization and Generation of Deployment Strategies for Charge Robotics' Solar Farm Construction Pipeline

Design Report

Prepared By:

Armita Khashayardoost, Katerina Vovk, Nathanael Kusanda, Arvin Bal

Team 2

Executive Summary

This Design Report presents our work with Charge Robotics in designing a model to optimize the placement and movement of their mobile factories, for deployment in their solar farm projects. The engineering opportunity is first established in an impact analysis identifying its potential societal and environmental impacts. The opportunity is then codified in a set of design and model quality requirements, focusing our design work towards the optimization of project time and cost. In response to this opportunity, we detail our recommended design, which is an Optimized Deployment Strategy Generator informed by Mixed-Integer Programming. Our design is evaluated against the requirements through comparison with Charge Robotics' existing process for resource allocation for a given test case, and is found to result **in a reduction in project completion time of one day, and a cost reduction of over $8,500 USD.** Our model is then used to investigate the trade-off between time and cost of a project, based on the number of vehicles and factories deployed, providing useful insights for Charge Robotics' future planning. Finally, we recommend potential future enhancements to our model, including the use of route weighting to consider variations in terrain.

Table of Contents

Introduction

This report outlines our design work towards the optimization of Charge Robotics' robotic system for solar farm construction. While the current system improves efficiency, challenges remain in minimizing construction time and optimizing factory placement. Our proposed model focuses on strategic factory deployment to address these challenges. The report evaluates key design decisions and assesses the solution's effectiveness, providing a pathway for Charge Robotics to enhance operations and scale their impact in the renewable energy sector.

Background Information

Context on Charge Robotics

Charge Robotics is a rapidly growing company based in Oakland, California, specializing in the automation of solar farm construction. With solar energy emerging as one of the cheapest and most scalable forms of power generation, demand for solar installation has immensely increased. However, large-scale solar projects face significant challenges due to labour shortages and logistical constraints. Charge Robotics is addressing these challenges by developing autonomous robots and mobile factories that improve the most labour-intensive and time-consuming aspects of solar farm construction.

The company's current product, the Sunrise system, is a mobile robotic construction factory designed to automate the assembly of key solar farm components. The system uses robotic arms to assemble solar modules, brackets and fasteners into solar bays, each consisting of 7-8 modules. Once assembled, these solar bays are transported to designated areas on-site using delivery vehicles, where workers install them into rows of solar arrays. Although some manual labour is still required for assembling and transporting the solar bays, Charge Robotics' introduction of automation makes large-scale solar farm deployment much more efficient. This innovation allows companies to meet rising demands while reducing both labour costs and project timelines.

Based on our meetings with Charge Robotics (Appendix A), their current deployment process begins with positioning up to three portable robotic factories on a solar farm site, where the assembly work is handled. The potential factory locations and delivery vehicle routes are pre-determined with consideration for terrain, regulations, and access to roads. The typical workflow then involves:

1. Robotic arms within the factory assembling components into solar bays, supported by three workers – one managing the software, one loading parts into the factory, and one loading pallets of solar modules.
2. Once the bays are assembled, they are loaded onto delivery vehicles and transported to the field, where three workers install them into rows of solar arrays. There are currently four delivery vehicles per factory each with a dedicated driver.
3. The factory can only hold one completed solar bay at a time, so the vehicles must rotate efficiently around the field to maintain a steady workflow.

Despite the improvements brought by automation, several logistical complexities remain that can be further optimized. This includes managing the movement and placement of the robotic factories to minimize construction time and ensure efficient delivery of solar bays to workers. As Charge Robotics scales up and invests in more equipment, optimizing factory placement will be crucial for completing projects on time, accounting for potential delays, and maintaining a competitive edge in the growing solar energy market.

Impact Analysis

Charge Robotics' contribution to the energy transition is exemplified in their deployment of mobile factories to aid the installation of solar farms. By leveraging automation, they have streamlined the construction pipeline significantly, as their factories enable bays of solar modules to be assembled on-site and installed upon assembly, greatly reducing procurement and transportation costs. This has enabled solar power to become cheaper, improving the viability of grid-scale renewable energy as an energy supply that can uphold society, while being a clean alternative to fossil fuels. Building upon this process, the opportunity arises to plan more concretely the deployment of mobile factories, optimizing their placement and movement to further expedite the construction process. In this section, impacts stemming from our work will be identified and then integrated into our expanded problem definition below, which includes strategies and principles for risk mitigation.

Safety Impacts and Analysis

A key component of the optimization model is the movement of factories to locations that most effectively reduce construction time. This new process for Charge Robotics introduces additional safety risks that have not been previously considered. To begin, each movement of mobile factories must adhere to ISO safety standards during de-installation, transport by trucks, and re-installation, similar to the standards followed currently during the initial setup of these factories [1]. Additionally, relocating factories results in more frequent changes to vehicle delivery routes and a more efficient system for collecting and installing bays. Together, these factors increase the overall frequency of vehicle movement across the solar site, raising the likelihood of vehicle collision and hazards. However, the adverse is also a possibility as clustering the location of factories throughout sites can allow vehicles routes to be delineated, reducing interference and collision risks.

It is important to note that certain hazards associated with movement patterns are not accounted for within the current scope of the model. We assume that the provided site plans have already undergone a safety analysis to identify and mitigate any historically observed risks or known hazardous movement patterns. This limitation must be acknowledged when using any results produced by our model.

Ethical Impacts and Analysis

The primary ethical consideration in introducing optimization into Charge Robotics' current deployment process is its potential impact on human labour. The model is designed to improve efficiency by determining strategic factory placements to reduce construction time. Since these vehicles and factories are operated by personnel, any change in their demand could affect job availability. Additionally, the relocation of factories requires additional personnel, potentially increasing labour needs in certain

situations. Overall, while automation can result in reduced labour needs, Charge Robotics' current process still heavily relies on human labour for tasks such as bay installation, factory movement, and operational oversight.

Environmental Impacts and Analysis

Optimizing construction time and resource allocation can significantly reduce emissions from vehicles and factories that operate on diesel fuel. By minimizing idle time and delivery routes, the optimization model can lower the carbon footprint for solar farm installations. However, if the model suggests frequent movement of factories, there may be an increase in fuel use or electricity consumption due to the carbon-intensive nature of these movements, potentially offsetting these environmental benefits. Furthermore, the construction and relocation of factories to various locations around the site may disrupt local ecosystems, affecting wildlife and vegetation. To mitigate these negative environmental impacts, Charge Robotics may need to conduct additional environmental impact analyses to ensure compliance with regulations, including the Chapter 7 Environmental Stewardship provisions proposed by California legislation regarding environmental standards in outdoor construction [2].

Societal Impacts and Analysis

By enhancing the efficiency and cost-effectiveness of solar farm construction, the optimization model can support the wider adoption of solar energy for large-scale electricity production. Increased utilization of solar energy not only supports the transition to renewable sources but also aligns with American targets aimed at reducing carbon emissions by 50% below 2005 levels by 2030 [3]. However, the greater reliance on automation and optimization may decrease the demand for human labour, potentially impacting job availability. In energy-producing regions around America, many communities prioritize energy sources that generate local employment and foster economic growth, which has historically contributed to the dependence on fossil fuels in the United States [4]. Without similar benefits, public perception of renewable energy projects may suffer if communities view these initiatives as favoring companies at the expense of local workers.

Definition of Engineering Problem

Building upon Charge Robotics' established solar project pipeline, our value proposition consists of developing a model that **generates a plan for the deployment of mobile factories**, optimizing the allocation of available resources to **reduce project completion time, with consideration of cost,** for any given Charge Robotics solar project.

Strategies for Mitigation of Impacts

In response to the identified environmental impact of large-scale construction, optimizing for project time allows resources to be more efficiently allocated, reducing the environmental impact of their operation. With the bottleneck of a factory output buffer zone, expediting the turnover from bay assembly to delivery reduces the time during which a factory or vehicle remains idle (See *Objective 1.1 Buffer Zone Delays* and *1.2 Vehicle Utilization)*, unable to assemble a new bay while still running and producing carbon.

However, a trade-off exists, in which a decision advised by the model may further reduce project time, but with diminishing returns or unforeseen impacts. As identified above, this is especially pertinent in the movement of a factory during the project, which may be advised despite the full day typically required to do so, as it may allow for the remainder of a project to be completed more quickly *(Objective 1.4 Optimal factory relocation frequency)*. As moving a factory is an extensive endeavor involving multiple vehicles and individuals, unforeseen safety hazards may be imposed that are not captured in the model. If one factory is being moved while another is operating as usual, the path of movement for one factory may coincide with a vehicle servicing the other, posing the risk of a collision.

While it is difficult to predict such coincidences precisely, this risk can be mitigated by communicating explicitly the positions of other factories during this precarious transition. As discussed in our **Proposed Conceptual Designs**, our Sequential Objective Function can be used to identify the conditions of the entire site based on the status of the buffer and available vehicles during a given timestep, enabling a better understanding of this safety risk. The need for model transparency and reporting is codified in *Objective 5.1 Client-side usability,* as discussed in our **Revised Requirements**. This information ought to be supplemented by an environmental impact assessment of the route to be taken for factory movement. As a general principle, information on conditions on the ground should always take precedence over instructions advised by the model, and it should always be at the discretion of the workers to judge outputs from the model.

The risk of collision is also posed by vehicles servicing different factories, and whose delivery routes may interfere. In theory, systematically selecting the placement of factories effectively allows for bays to be clustered according to their assigned factories. The resulting delineation between clusters should reduce the likelihood of collision between vehicles servicing different clusters. These clusters ought to be visualized and communicated to the workers on-site, so that they can be adhered to during bay delivery to mitigate the risk of collision.

A societal impact we identified above was the displacement of human labour stemming from automation. In our case, labor is still required to monitor and occasionally operate the mobile factories, and is required for the delivery and installation of assembled bays. Even so, we believe strongly in including the human in the loop: information captured on-the-ground by workers, especially with regards to environmental conditions, is valuable and cannot fully be captured by any model. The model should primarily function as a tool for engineers and technicians on-site, communicating its decisions as clearly as possible to better facilitate their planning and enable their judgements on whether or not said decisions ought to be accepted.

Scope Definition

Scope Tier	# Mobile Factories	Vehicle Localization	Factory Movement
Scope 1	Single	Localized	No
Scope 2A	Single	Localized	Yes
Scope 2B	Multiple	Localized	No
Scope 3	Multiple	Delocalized	No
Scope 4	Multiple	Delocalized	Yes

Table 1. Characteristics of Tiered Scope

As this is a multi-faceted problem, organizing our design into tiered scopes enables our solution to be progressively constructed. Accumulated work is checkpointed at each incremental tier, before expanding. If issues in implementation arise, referring to previously completed tiers allows issues to be clearly identified. Additionally, scope tiers enable iterative design by allowing individual tiers to be redesigned without affecting completed solutions from previous stages . Furthermore, optimization methods that solve simpler scope tiers may not be adequate in solving more complex ones, thus allowing for a clearer understanding of the varying complexity within this optimization.

Three variables are considered when defining tiers: 1) the number of factories, 2) the consideration of vehicle localisation, and 3) the movement of factories during construction. Moving from a single factory to multiple factories complicates the problem, as a mechanism has to be implemented to assign bays to each factory. Additionally, time is less easily estimated, as the factories run in parallel, and time steps for the multiple factories must be tracked simultaneously. Vehicle localisation follows: in the case of a single factory, vehicles can only return to a single location after installing a bay, but for multiple factories, the vehicles should be considered to be floating, as they can service any factory. Deciding which factory a vehicle should service introduces another element of complexity, as does the consideration of factory movement, as our model must determine when it is advisable to do so.

Design Requirements

The Design Requirements have been adapted from the Project Proposal, to the Design Review, to the final Design Report. The updated requirements are shown in Table 2 below, with the following colour-coding:

- The Green sections indicate the requirements we are explicitly designing for in our conceptual designs.
- The Yellow sections are requirements we are either implicitly including or will be considered in further iterations of design.

Design Requirements			
Objectives	**Metric**	**Criteria**	**Constraints**
1.0 Optimize Construction Time			
1.1 Buffer Zone Delays: As soon as the bays are assembled and placed in the buffer zone, a vehicle should be prepared to pick it up for transportation.	Buffer zone wait time (s)	Minimal as possible	N/A
1.2 Vehicle Utilization: Vehicles should be idle for as little time as possible during construction.	Vehicle Active time (s)	Higher active time is preferred	N/A
1.3 Optimal Vehicle Trip Distance: Route plans should be selected such that vehicles can complete installation and return to collect the next bay without bottlenecking the buffer zone.	Ratio of assembly time to average vehicle return frequency	As close to 1:1 as possible	N/A
1.4: Optimal Factory Relocation Frequency: There may be situations where moving the factory (requires one day) to a new location would save time by enabling the vehicles to travel shorter distances for each trip. Determining the optimal frequency based on the distance of the planned installation from the factory would minimize the construction time.	Ratio of time saved from movement to time required to move	As close to 1:1 as possible (i.e. a vehicle should return as soon as a new bay is assembled)	N/A
1.5 Optimal Factory Placements: Factory should be placed in viable locations that minimize construction duration.	Construction duration (hrs)	Minimal as possible	Must be selected from the provided set of viable factory locations
1.6: Deployment Robustness: Design should be robust to unexpected changes, including vehicle/factory outages.	Risk frequency and severity	Minimize risks frequency and severity	N/A
1.7: Construction Completion Date: The overall construction completion date must be within the deadline, and ideally as short as economically and practically possible.	Duration to Construction Completion (days)	As soon as economically and practically possible is preferred	Must not exceed specified deadline
2.0 Optimize Economic Costs			
2.1: Optimize Number of Factories Deployed: To reduce cost of depreciation, fuel, emissions, and labor for the factories, the number deployed should be minimized such that timing requirements are met.	Cost of factory Deployment ($USD)	Minimum such that timing requirements are met	Maximum number factories provided by Charge Robotics
2.2: Optimize Number of Vehicles Deployed: To reduce cost of depreciation, fuel, emissions, labor and to reduce congestion, the number deployed should be minimized such that timing requirements are met.	Cost of Vehicle Deployment ($USD)	Minimum such that timing requirements are met	Maximum number vehicles provided by Charge Robotics
2.3: Standard Construction Hours for Labour Force: Reduce the hours required for overtime and weekend shifts (1.5x pay).	Hours on Weekends and Overtime (hrs)	As low as possible is preferred	Must be within state regulated Labor Laws of construction site

Table 2. Design Requirements

Model Quality Requirements			
Objectives	**Metric**	**Criteria**	**Constraints**
3.0 Scalability			
3.1 Applicability to Other Sites: The model should be scalable to more/larger projects and to situations in the future where the Client has more vehicles/factories at their disposal.	Range of site plans/size the model can handle	Largest range practically possible is preferred	N/A
4.0 Versatility			
4.1 Customizable Parameters: By allowing the parameters to be changed, when the company needs to make adjustments to installation times, travel times, or other factors, the model should be robust to these changing parameters.	Number of adjustable parameters	More adjustable parameters is preferred	N/A
4.2 Availability of Mathematical Models: The model should be provided with the solution such that it is easy to understand for the clients, modifiable, and an accurate representation of the costs involved.	Whether or not mathematical models are available	N/A	Mathematical models must be available to the Charge Robotics once complete
4.3 Multiple Solution Outputs: The model should output multiple viable solutions for a given situation, allowing the client to select the best choice based on their situational needs.	Number of Solution Outputs	*Subject to Charge Robotics input	N/A
5.0 Usability			
5.1 Client-side Usability: Once completed, it should be easy for the client to enter the inputs (site plan, deadlines, factory/vehicle availability) into the model and receive their recommendations.	Direct Client feedback	As usable as possible is preferred	Client must be able to use the model with a high level of effectiveness
6.0 Reporting Requirements			
6.1 Final Report: The model should be accompanied by a comprehensive report outlining the problem framing, model framework, and a worked example based on an existing solar site.	Quality of the final report	Higher quality report is preferred	Must include comprehensive report

Table 3. Model Quality Requirements

One of the main modifications to the requirements from the Design Review to the final product has been the inclusion of *2.0 Optimize Economic Costs*. While we had scoped this out during the Design Review, after conversations with Charge Robotics, we have decided to include it back in the scope. We used parameters of the cost from Charge Robotics, such as labour costs, depreciation, and deployment costs to estimate the total cost for a given scenario. The cost estimate will allow Charge Robotics to make informed decisions about the trade-offs between cost and completion time when deciding on the number of factories and vehicles to deploy for a site.

Additionally, as we expanded our scope to include the case of factory relocation, we have been able to optimize requirement *1.4: Optimal Factory Relocation Frequency* by determining how frequently the factories should be moved to minimize the construction time. We have also included all of the Model Quality Requirements into our final product, and we have validated the requirements with Charge Robotics. These will be further elaborated on in the **Verification of Recommended Design** section.

Recommended Design

Framework of Proposed Design Solution: Optimized Deployment Strategy Generator

Our design solution comprises three parts: a sequential objective function, a Mixed Integer Programming (MIP) optimizer, and a cost estimator. Over the course of this section, details of each component will be described, with certain design decisions being highlighted and described. The three parts are tied together as follows:

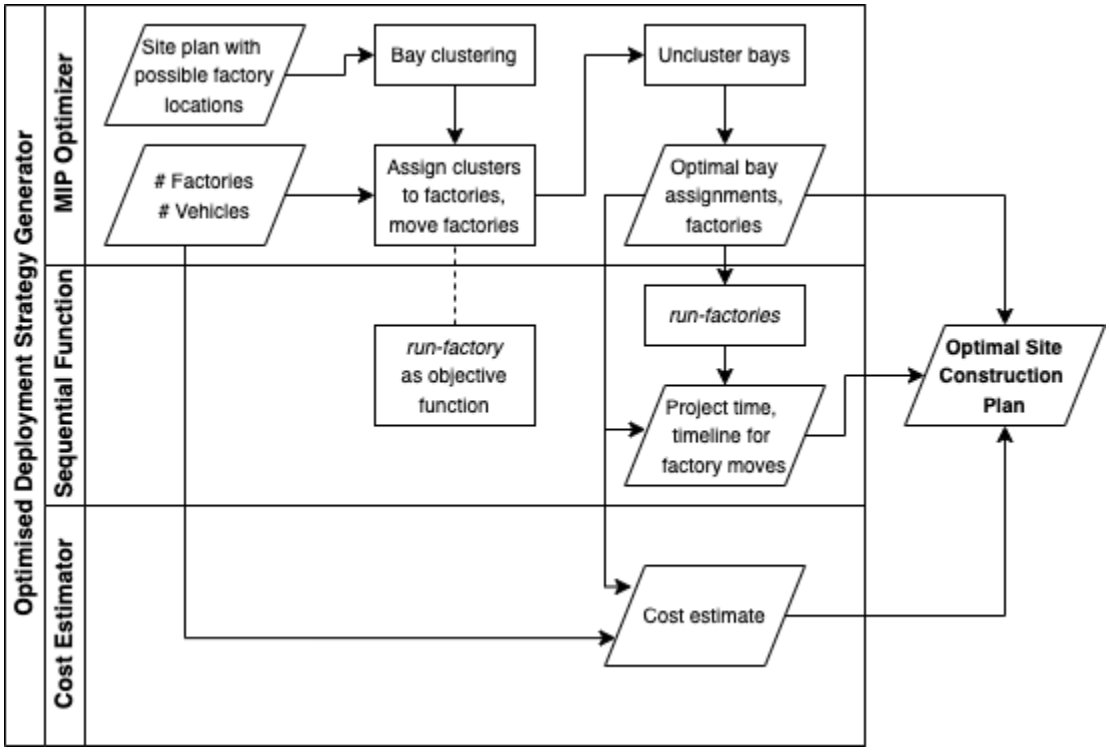

Figure 1. Framework of Optimized Deployment Strategy Generator

Our generator takes as inputs a maximum available number of factories and vehicles, and a solar farm site plan containing possible locations for factories to be placed. These inputs are fed into our MIP optimizer, which utilizes a simplified version of the sequential objective function, *run-factory,* to determine optimal assignments of bays to each factory, and optimal locations and movements for each factory. After an optimal bay assignment and set of factory locations / movements is determined, the holistic version of the sequential objective function, *run-factories*, is used to produce a visual site construction plan depicting bay assignments, factory locations, including a timeline of each factory's movement. This function is also used to estimate the total project time, which will be outputted along with an estimated project cost.

Instructions for use of Optimized Deployment Strategy Generator

The user should provide the following inputs into the generator:

- *Site plan:* A list of all bay coordinates in a solar farm site to be constructed
- *Possible factory locations:* A list of viable coordinates at which a factory may be placed
- # Factories and # vehicles available

For a given set of available resources, the generator will determine the optimal resource deployment strategy based on time, and produce a cost estimate. As such, the generator can be used to examine trade-offs in resource deployment, which is elaborated upon in the ***Investigating Cost-Time Trade-off of Resource Allocation*** section below. For instance, if three vehicles are available, the user may start by inputting one vehicle into the generator. If the determined optimal construction plan provides a time estimate that exceeds the user's deadline, the generator ought to be re-run with an additional vehicle, until the deadline is met. Additionally, the deployment of additional resources may in fact result in a site plan that is not only faster, but cheaper, due to the cost of a project being correlated to an extent with the time over which each resource is deployed. A similar iterative process can be followed for the number of factories available.

The optimal site construction plan is thus determined for a given set of available resources. The later section on ***Examples of Deployment Strategies Generated by Proposed Design*** exemplifies the visual depiction of the optimised site plan, which color-codes bays based on the factory it is assigned to, with different shades of the same color depicting different locations for the same factory. The legend provides information on where and when each factory is deployed or moved.

Details of Sequential Objective Function

In addressing requirement *1.0 : Construction Time Optimization*, the first step lies in the creation of an objective function, i.e. a function which quantifies the solar site construction time and whose value is to be minimized over the set of feasible factory locations. The simplest form of this function represents **Scope 1,** for a single factory without movement. This version is named *run-factory,* and is depicted in Appendix B1.

As seen above, time is treated as a discrete variable akin to minutes, and the function returns the amount of discrete time steps required to build the solar site. While each integer time step corresponds to a minute, this output should be scaled to enable measurement of requirement *1.7 Construction Completion Date* and ensure it meets the deadline constraint. By depicting construction in time steps, the conditions of the site can be tracked through the various queues and lists and reported, enabling mitigation of safety impacts identified in our **Impact Analysis.**

This version of the function is used in the MIP optimizer on a single-bay basis to determine the allocation of a bay to a factory. As feeding a single bay into this function flattens its representation of requirement *1.1 Buffer Zone Delays,* which would only arise when several bays begin to bottleneck a factory due to the absence of vehicles to relieve the buffer, and the installation and assembly time is constant, the function essentially estimates the bay delivery time. However, delivery time is still related to buffer delay, as the longer it takes for a bay to be delivered, a vehicle would be unavailable for a greater duration, thus inhibiting the relief of other bays in the buffer.

The bay installation time **T_install**, assembly time **T_bay**, and time required to move a factory **T_move** are assumed to be constant, based on parameters provided by Charge Robotics. The third assumption is safe to make since **T_move** is a few time increments greater than the time required to build a bay. These assumptions allow for the treatment of building times as a linear variable, dependent on the distance from bay to factory. The Manhattan distance is used as a heuristic for measuring delivery distance. While this

provides a simplification to requirement *1.3 Optimal Vehicle Trip distance* where the vehicle routes and traffic are ignored by the design scope, it is justified, as the progressive completion of rows of solar bays will inevitably restrict vehicle movement to the X and Y direction when delivering bays to locations further away. However, the objective function still leaves room for flexibility in selecting a different distance heuristic if required in later project stages.

Using this simplified function allows for the optimization problem to be solved on a single-bay basis, and for the site construction plan to be constructed progressively as such. However, to more accurately estimate project time, a more holistic function is required. This is specified as *run-factories,* and is shown in Appendix B2.

This version of the function is used after an optimal site construction plan is produced, to accurately estimate the completion time of a project, and determine the time steps at which a factory is moved. By tracking multiple moving factories and the progress of construction of assigned bays, and depicting the vehicles as delocalized across these factories, this function captures the required characteristics of **Scope 4.** Building upon the base code of *run-factory* to capture these nuances required several modifications, which are highlighted below.

Firstly, to track several factories, lists were converted to dictionaries of lists, indexed by the factory number. At each time step, each factory's remaining planned bays, bays in the buffer, and installed bays are tracked. Secondly, the new dictionaries **factory_start_time** and **factory_finish_time** track the timeline of each factory's movement through its respective locations. Thirdly, to capture the delocalization of vehicles, cars are tracked with lists that capture the time at which a vehicle finishes delivering its current bay, the presently available cars, and the current bay location the vehicle must deliver to.

Delocalization is further captured in the delivery distance being computed as a sum of the distance from the vehicle's current location to the factory it is to relieve, and the distance from that factory to the new bay location. This is in contrast to *run-factory,* where the vehicle is assumed to begin and end its journey at the same factory. If a factory has bays to be relieved from the buffer and vehicles are available to deliver it, the closest vehicle is determined and called upon. To ensure no single factory sweeps newly available cars, which would be the case if the same factory was always tracked first at each time step, the order in which the factories are tracked is sorted based on the number of available bays remaining. This ensures an even distribution of vehicles across the site plan.

The input to *run-factories* is a dictionary of factory and bay assignments, and corresponds to the output of the MIP optimizer. For each factory key, a nested dictionary is contained representing all factory locations, and lists of bays for each location. If the list is non-empty, the factory will be moved to that factory location at some point in the construction process. A simplified depiction of the input is included in Appendix B2.

Details of Cost Estimator

The cost estimator is called in *run-factories* to produce a project cost for a given site construction plan. For both factories and vehicles, tt comprises the following components:

- Flat cost of deployment
- Linear depreciation
- Required labour

The flat costs depict the fixed rate of deploying a factory or a vehicle at a site. On the other hand, the linear depreciation captures the hourly cost of using these resources, due to their deterioration over time. Both values were provided by Charge Robotics, and can be adjusted to depict future revisions of these costs. For a given factory, three people are required to monitor and coordinate bay assembly, and three people are required to install the assembled bay on the field. For a given vehicle, one person is required to drive the vehicle. Labor costs are thus computed using the project time and a standard payment of $25 USD/hour. The code for the cost estimator is shown in Appendix B3.

Implementation of Mixed Integer Programming

In modeling factory selection as an MIP optimization problem the following decision variables are defined:

- X[i][j][k] - “service bay”. In this binary array, a value of 1 is assigned if bay i is serviced by factory j at factory location k, 0 otherwise.
- Y[j][k] - “move factory”. In this binary array, a value of 1 is assigned if factory j is moved to factory location k, 0 otherwise.
- max_time - a continuous variable which keeps track of the site construction time given an assignment of bays and factories.

Our objective function minimizes the max_time variable, which is a continuous function of the decision variables, as conditioned by specific constraints.

This particular selection of decision variables enables the Scope 3 tier to be covered as both factory movements and the presence of multiple factories are accounted for. MIP allows for optimization of a single objective at a time. As minimizing project time has been specified by Charge Robotics as the primary objective, we have chosen to prioritize this in our model construction. The objective function directly calls run_factory for each bay location. This is a good proxy for the total site construction time but does not take into account the panel buffer time. The code excerpt in Appendix B5 contains the MIP implementation.

To overcome challenges relating to code efficiency a clustering technique is employed. Prior to application of optimization, the panels are grouped into clusters, and the objective function is called from the center of each of the clusters. This significantly reduces code runtime, in the employed test case, the number of panels is reduced from roughly 20,000 to 1,000. After the optimization is performed, the panels are unclustered to their original state. Appendix B6 shows the clustering function, where k-means is employed for cluster creation.

Examples of Deployment Strategies Generated by Proposed Design

The proposed design was applied to a dataset provided by the client under various input factory and vehicle resources.

Figures. 2A-D presents examples of deployment strategies generated by the proposed design. These results follow a certain degree of intuition whereby bays are grouped together based on location and the size of the groupings is relatively even. However, there are also some areas where the bay assignment boundaries are not polygonal, something a purely human strategy would never suggest.

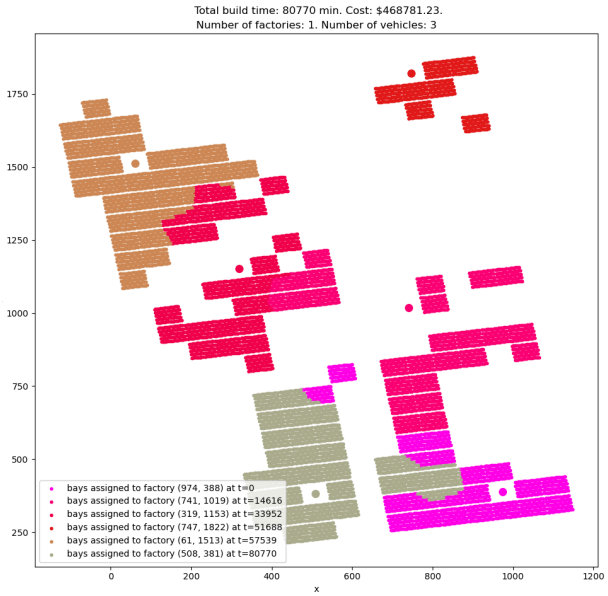

Figure 2A. Strategy for 1 factory, 3 vehicles

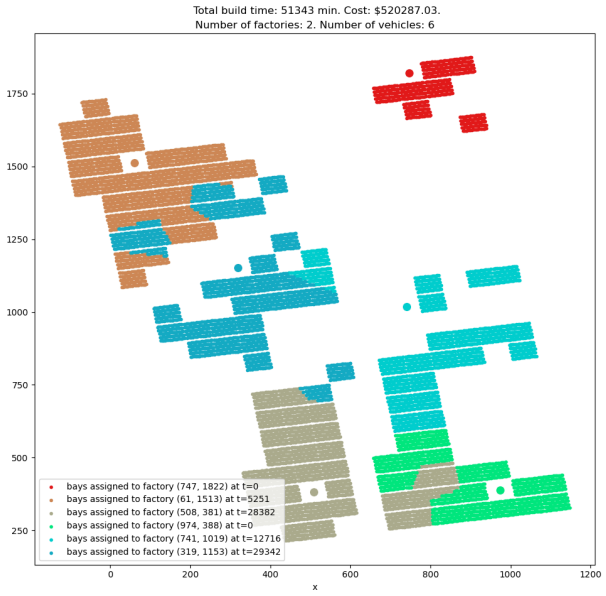

Figure 2B. Strategy for 2 factory, 6 vehicles

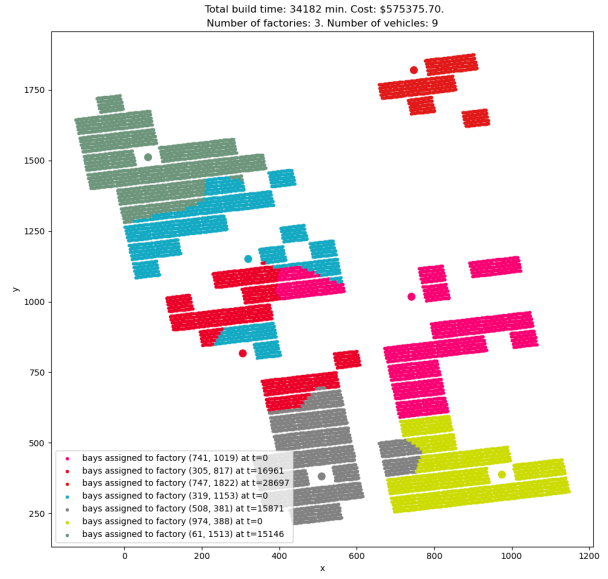

Figure 2C. Strategy for 3 factories, 9 vehicles

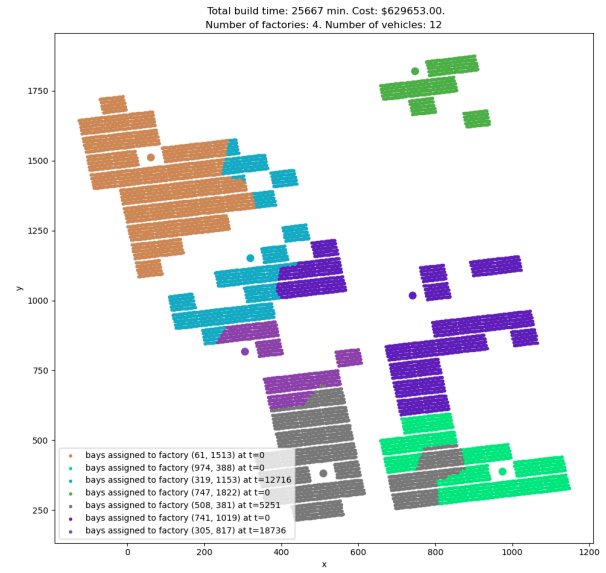

Figure 2D. Strategy for 4 factories, 12 vehicles

Validation of Recommended Design

Validation of Sequential Objective Function with Hand Calculations

To validate the *run_factory* and *run_factories* functions used as the sequential objective function, hand calculations were performed on five basic test cases. The cases were structured as follows:

- **Cases 1 and 2**: Evaluate *run_factory* to identify the best factory location with three possible factory options and three bays to construct. Case 1 assumes one vehicle, while Case 2 assumes two vehicles.
- **Cases 3 and 4**: Evaluate *run_factory* to identify the best factory location with three possible factory options and six bays to construct. Case 3 assumes one vehicle, while Case 4 assumes two vehicles.
- **Case 5**: Evaluate *run_factories* to construct three panels initially, then determine whether moving the factory to a second location is more efficient for building the remaining three panels with two vehicles.

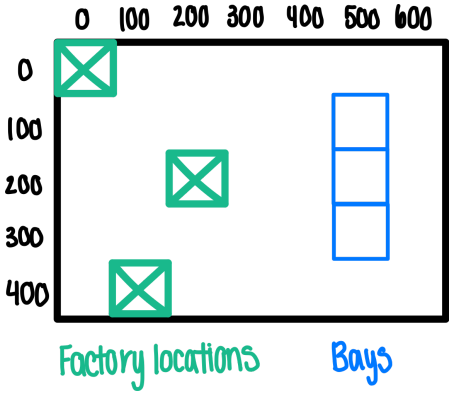

Figure 3A. Case 1 and 2 Setup

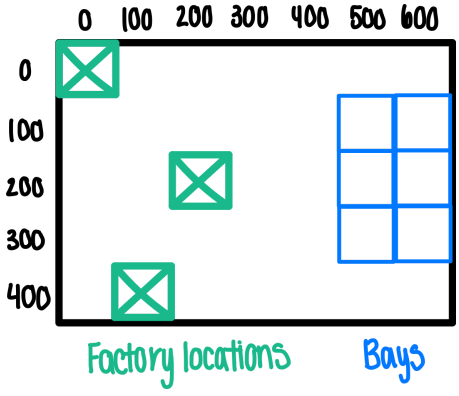

Figure 3B. Case 3 and 4 Setup

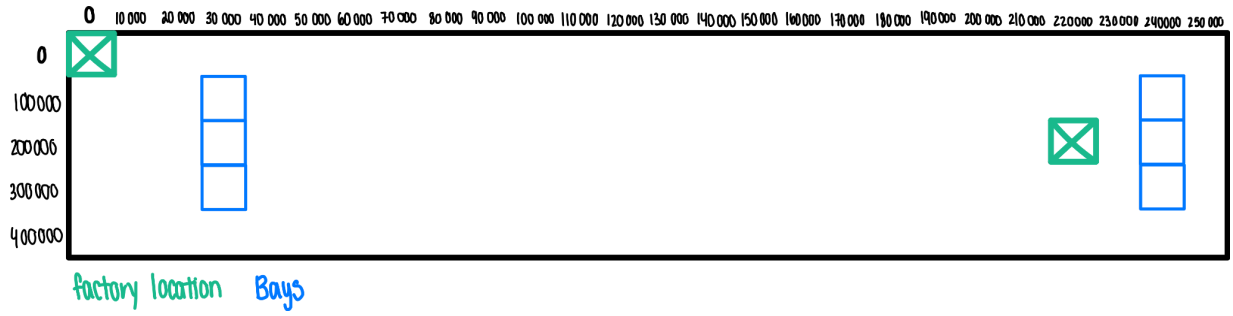

Figure 3C. Case 5 Setup

Hand calculations for each case followed a sequential approach, analyzing the time required to complete each step. Detailed calculations are provided in Appendix 6.

The result for each of these hand calculations matched the results outputted from *run_factory* and *run_factories* providing a base sanity check. For case 1, it was expected that placement of the factory at (200,200) would result in the lowest time 29 minutes. For case 2, with 2 vehicles, the factory at (200,200) is still expected to have the lowest time but now at 23 minutes. For case 3, the factory at (200, 200) still

had the lowest time with 1 vehicle resulting in 57 minutes. For case 4, the factory at (200,200) and (100,400) would result in the same time of 40 minutes with 2 vehicles. Lastly case 5, with the factory movement and 2 vehicles is expected to have a time around 2486 minutes.

The hand-calculated results aligned with the outputs from the run_factory and run_factories functions, providing a basic sanity check:

- **Case 1**: The factory located at (200, 200) resulted in the lowest time, 29 minutes.
- **Case 2**: With two vehicles, the factory at (200, 200) again yielded the lowest time, 23 minutes.
- **Case 3**: With one vehicle, the factory at (200, 200) produced the lowest time, 57 minutes.
- **Case 4**: With two vehicles, the factories at (200, 200) and (100, 400) resulted in the same time, 40 minutes.
- **Case 5**: The factory moving from (0,0) to (220000,200000) with two vehicles resulted in a total time of approximately 2486 minutes.

These results confirm the validity of the *run_factory* and *run_factories* implementations for the tested scenarios.

Verification of Recommended Design

Justification of Recommended Design Against Other Options

Over the course of our design process, several design options were considered before converging on our final design. Leveraging the tiered scope definition identified in our ***Scope Definition*** section, we were able to systematically assess which solutions were viable, based on their capacity to capture successive scope tiers. The following table summarises our assessment of different options:

Design Option	Feasibility	Advantages	Drawbacks	Max Scope Achieved
Standard MIP Model (k-Median, k-Center)	High feasibility as we have clearly defined objectives and constraints. However, when the solution space gets large and highly constrained, finding an optimal solution may become computationally prohibitive.	Extensive documentation in literature Can capture multiple factory cases Guarantee of a unique optimal solution	Time is optimized only by proxy of distance Requires further augmentation to consider factory movement (*Scope 4)* Proxy does not capture impacts of vehicle delocalization	Scope 2B
Heuristic Method	High feasibility, allows for quick solutions and flexibility. Can adapt well to various constraints and provide	Direct application of **sequential objective function,** which can be refined to capture vehicle delocalization	Requires further augmentation to consider factory movement (*Scope 4)*	Scope 3

	satisfactory solutions in a shorter time frame, which is beneficial for real-world applications where exact optimization may not be feasible.	Easily captures multiple factory cases	Does not guarantee an optimal solution and may get stuck on a local optima	
Dynamic Programming	DP is applicable when the problem exhibits overlapping subproblems and optimal substructure properties. This means that the solution can be constructed efficiently from solutions to smaller instances of the same problem. Due to the lack of similar applications of DP in the literature, problem formulation would have to be novel.	Avoids redundant calculations through memoization Captures sequential nature of problem Breaking problems into subproblems may allow for either a single factory to be tracked through different locations, or multiple factories to be tracked at static locations.	Iterative nature can lead to large computation times and high memory usage DP works best for problems with the right properties (optimal substructure and overlapping subproblems). Breaking our problem into subproblems in two axes of complexity (multiple factories, dynamic locations) may not be viable.	Scope 2A or Scope 2B
Customized MIP	Existing MIP problems can be built upon to capture unique elements of our problem. Use of clustering can reduce computation time significantly.	**Sequential objective function** can be used directly in optimization. Can capture both axes of complexity (multiple factories, dynamic location)	Only a simplified version of the **sequential objective function** can be used directly, however, a holistic version can be applied following optimization to assess solutions generated.	Scope 4

Table 4. Design Options Comparison Matrix

Our chosen solution is labelled as the Customized MIP, and as described in Table 4, is able to capture the full extent of the problem as specified in Scope 4. That being said, our recommended design leverages elements of other design options. Standard MIP Models were built upon, with optimization variables expanded to capture specific nuances of our problem, such as by expanding the X matrix typically found in resource allocation problems in the literature by adding a third dimension to capture factory movement through different locations. Additionally, the k-Means problem formulation was drawn upon in our model to cluster bays and expedite computation prior to optimization. Furthermore, the sequential objective function was used directly in our optimization process, as per the Heuristic Method.

Testing Proposed Solution against Charge Robotics' Existing Process for Resource Allocation

For the test location site, we asked Charge Robotics how they would divide the site for construction deployment. Based on their evaluation, they determined that for a site of this size, the cost of moving a factory would outweigh the benefits. Therefore, the site was divided into regions with fixed factory placements as follows:

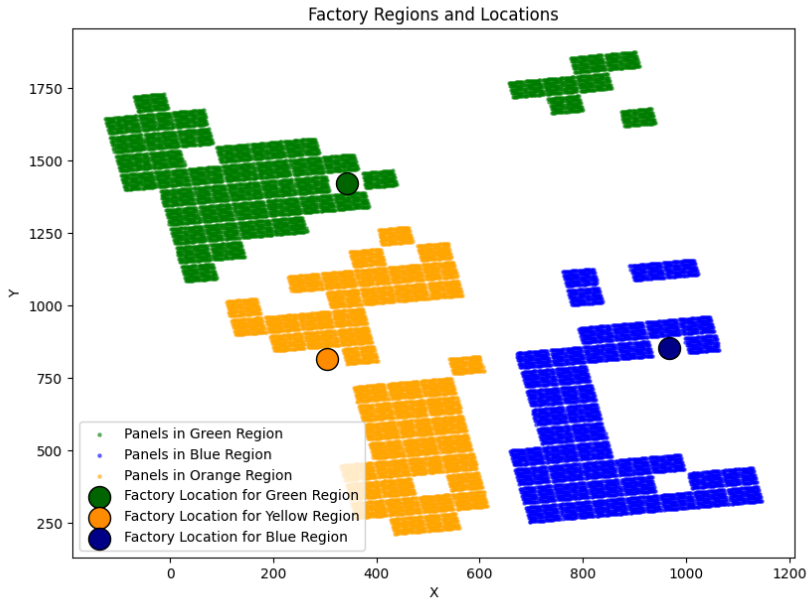

Figure 4. Charge Robotics Allocation

The deployment strategy for this site was to divide it into three regions, each assigned to one of three factories, with three vehicles per factory. All factories operated independently and in parallel to construct their designated regions. Using this deployment strategy, with a resource availability of three factories and nine vehicles, the time to complete construction was calculated to be 35,260 minutes (approximately 24 days) at a total cost of $583,883 USD. This approach did not involve any factory movements, and all regions were built simultaneously.

When applying our optimization model to the same site with the same set of resources, the total construction time was reduced to 34,182 minutes (approximately 23 days), and the cost decreased to $575,375 USD. This strategy is depicted in Figure 2C. **This resulted in a time savings of one day and a cost reduction of over $8,500 USD.** This difference demonstrates the potential of the optimization model to enhance efficiency by minimizing time, even for medium-sized sites. As site sizes grow, the optimization model is expected to achieve significantly greater time savings, particularly by strategically incorporating factory movements where cost-effective.

Investigating Cost-Time Trade-off of Resource Allocation

The MIP optimization models enable the user to select the right factory locations and movement frequency to minimize the deployment time, given a certain number of factories and vehicles. To enhance the decision-making process for Charge Robotics' deployment strategy, we used the MIP optimization and Cost Estimate models to create a cost-benefit visualization for various combinations of factories and

vehicles. The code iterates through each combination, using the MIP code to solve the minimum completion cost, and creates plots for each number of factories, showing how the cost and completion time change as vehicles are added. The plots for one, two, and three factories with one to nine vehicles are below:

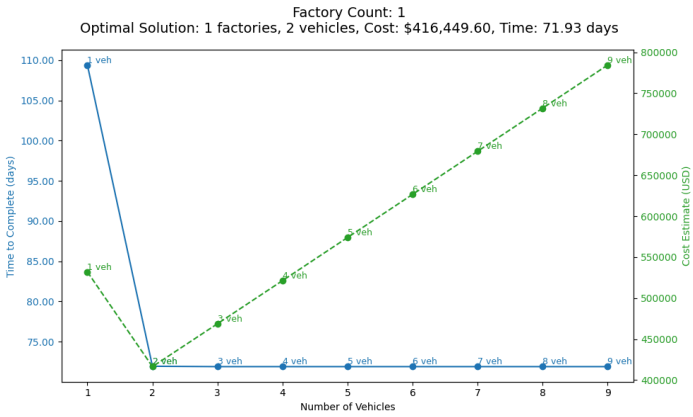

Figure 5A: Trade-off between time and cost for single factory case

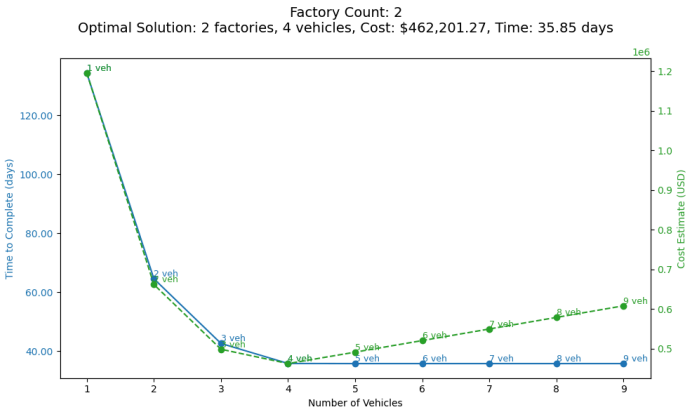

Figure 5B: Trade-off between time and cost for two factory case

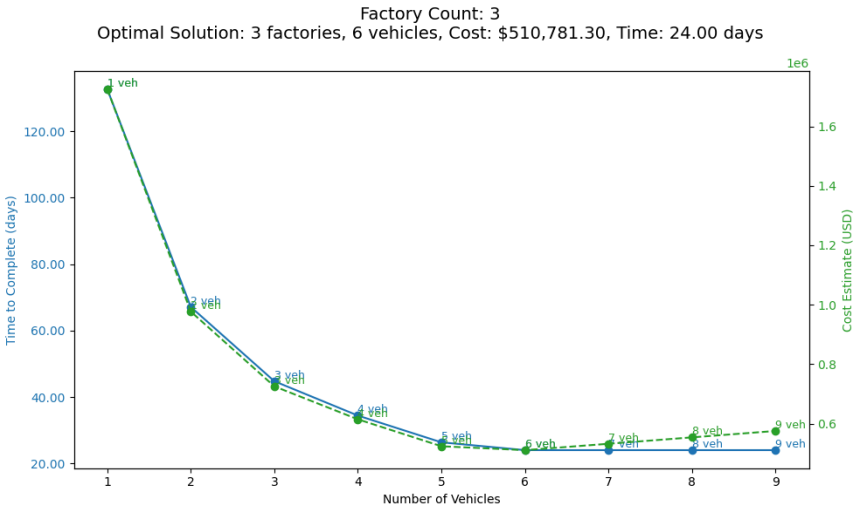

Figure 5C: Trade-off between time and cost for three factory case

The optimal deployment is selected by identifying the point where the cost is minimized, and due to the dependency of the time on the cost, the minimum time is closely correlated to the minimum cost. This causes the minimum cost scenario to simultaneously be the minimum time option, which is indicative in the plots. These optimal points are labelled on the plots. After this minimum, the completion time does not decrease with the addition of more vehicles, and this is due to the assembly time being the bottleneck after a certain number of vehicles.

In the plots for a given number of factories, we see a clear optimal deployment strategy based on the trend of having a clear minimum cost and minimum time scenario. However, these don't show the impact of adding factories, just adding vehicles. We expanded the model to plot combinations with different numbers of factories *and* vehicles all on one graph. Each colour represents a different number of factories, and the number annotated on the point indicated the number of vehicles for that scenario. A sample plot with combinations of up to three factories and up to nine vehicles is shown below:

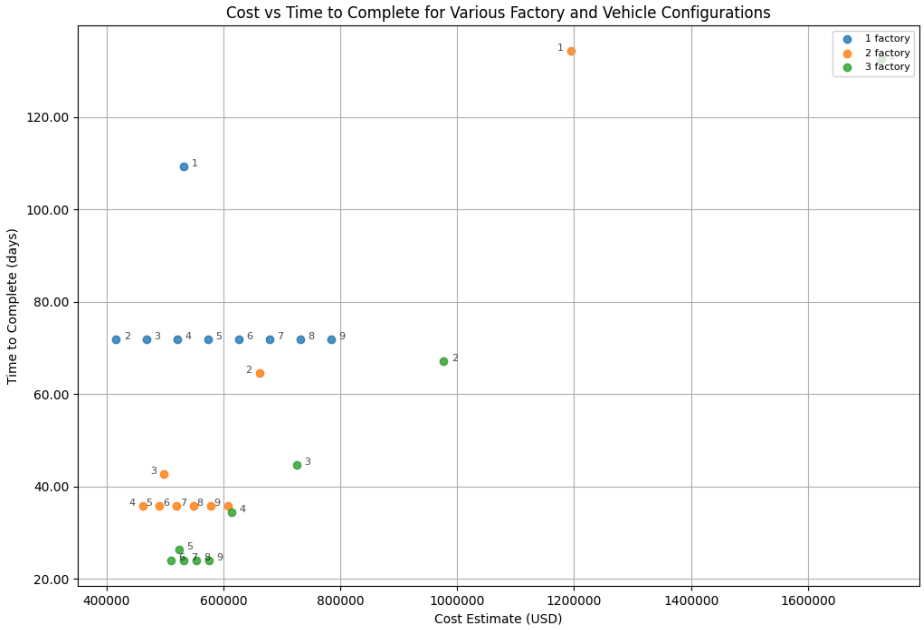

Figure 6: Trade-off between time and cost for various factory-vehicle configurations

Unlike the trends for a given number of factories, there is no clear optimum where both the cost and completion time are simultaneously minimized. We can see adding more factories naturally reduces the completion time, but the cost increases for the optimal case. For example, we see that the lowest cost option would be one factory, two vehicles – with a completion time of 71.93 days and cost of ~$416,229. On the other hand, the lowest time option is 3 factories, 6 vehicles – with a completion time of 24 days and cost of $510,781 (Appendix D). These results indicate that with 22.7% increase in cost, the completion time can be reduced by 66.7%. However, the deployment decision will be based on whether Charge Robotics values time or money for each use case. There may be cases where minimizing the cost or completion time might be more valuable than each other. The plot visualizes the estimations for both variables and allows Charge Robotics to make an informed decision when selecting their deployment strategy. If they purely want to reduce the costs they might choose less factories, and if they want to reduce the time they might select more factories. Either way, they can see the trade-off between time and cost estimates visually, informing their decision-making process.

Note that there are anomalies in the cost and time estimates for cases where there are less vehicles than factories, a case that is not realistic for Charge Robotics. The plot shows the time estimate for one factory and one vehicle is less than for two factories and one vehicle. This would not be the case in reality, and is caused by the sole vehicle moving between the two factories, increasing the time. Since Charge would not

deploy less vehicles than factories, and the time and cost estimates for this case are far from the optimal points, these outliers do not affect the outcome of the model.

Impact of Cost-Time Model on Requirements

While the Sequential Objective Function model works to satisfy requirements in '1.0 Optimize Construction Time', the Cost-Time model uses those results to meet the Requirements in '2.0 Optimize Economic Costs'. The model uses the Estimate Cost function to factor in the cost to deploy factories and vehicles, as well as the labour costs associated with construction, to provide the most economically effective deployment strategy. Charge Robotics can use the results of the Cost-Time model to evaluate which deployment strategy is required for each use case to meet both economic and construction time requirements.

Assessment of Model Quality Requirements

The optimization models fulfills the requirements in '3.0 Model Quality Requirements', including **scalability, versatility**, and **usability,** ensuring it can be integrated into Charge Robotics' design process after the model is handed off to them.

In terms of the **scalability**, the model is designed to be agnostic to the size of the site and various panel arrangements, allowing it to operate across a range of different projects and layouts. The only site input requirement is the map of the site and proposed factory locations. Based on any arrangement of potential factory locations and panels, the code uses the distances between the factories and the panels to compute the time required. One key strength of the model is its ability to scale up to larger sites with more panels, more factories, and more vehicles. However, as the size and complexity increases, the computational runtime of the optimization process naturally grows. Charge Robotics has verbally confirmed that longer runtimes for larger projects are acceptable, since the optimization only needs to be performed once and is not constrained by operation time. This capability ensures that the model remains effective and relevant as the company works on deploying their operations to larger, more complex sites.

Next, in terms of **versatility**, the model's parameterization of all components makes it highly flexible, and can be adapted to various updated requirements and parameters. All major inputs are clearly documented and commented within the code, allowing them to be easily adjusted upon receiving new information about realistic project schedules and costs. This also ensures the model can respond to specific project scenarios without extensive modifications. Key adjustable parameters include, but are not limited to:

- **Vehicle Speed:** Determines the time it takes for a vehicle to receive a bay, deliver to the installation location, and return to the factory.
- **Assembly Time:** Determines time for a bay to be assembled and ready to deliver to installation location.
- **Factory Movement Time:** Time it takes to move a factory to another location on site.
- **Cost Estimate Parameters:** Uses fixed and time-dependent cost rates of labour, depreciation, and deployment costs to estimate total cost for a given number of factories and vehicles.

The mathematical framework of the model, including objective functions and constraints are also accessible and modifiable within the code. The ability to adjust the model ensures that it remains versatile

and flexible for a wide range of use cases, and continually improves the model after the base model design is received.

Finally, the model is designed to be **usable,** ensuring it can be efficiently integrated into Charge Robotics' operational workflows. Through our weekly meetings with them, we have ensured the model is clear and understandable on their end, and that they will be able to use it with their current workflow. We consistently received positive feedback from Charge Robotics, validating that our model will be usable when they receive it. Inputting project data, including maps and other site-specific details is compatible with their current site maps, allowing them to seamlessly integrate our model into their workflow.

One of the most valuable features of our model is the ability to produce clear, visual, and quantitative results as outputs. Some of these include:

- **Colour-Coded Maps:** These maps show which bays are serviced by each factory visually, as well as where the factories move throughout the construction period
- **Animation of the Process:** We have produced animations of the construction process from the start to the finish so they can visually see the sequencing of the construction and how the factories move over time
- **Cost-Time Trade-Off Plots:** These plots allow Charge Robotics to visually see the effects of adding additional vehicles, and the cost-time trade-offs of adding more factories for deployment.

The combination of these visual outputs, alongside the quantitative results, ensure the model is not only usable for inputs, but that the outputs are clear and understandable to both technical and non-technical stakeholders. These features ensure the model is not only a powerful optimization tool, but also an effective communication aid, enabling Charge Robotics to communicate the benefit of their service to both internal and external stakeholders.

Conclusion and Recommended Next Steps

This Design Report outlines our progress in developing and refining an optimization model for Charge Robotics' mobile factory deployments. The proposed model generates an optimized layout based on the number of factories, the total number of vehicles available, and potential factory locations. While the primary focus of the optimization algorithm is minimizing construction time, a cost-time trade-off analysis is also conducted to identify an ideal layout that balances both objectives effectively.

Several assumptions were made within the current scope to simplify the problem, which can be expanded upon to enhance the model's capabilities. Future improvements include:

- **Road Mapping Integration**: Incorporating detailed road mapping into the optimization model to account for more accurate travel distances and site-specific constraints.
- **Dynamic Vehicle Allocation**: Currently, vehicles are localized to specific factories during optimization, with delocalization applied only during the time estimation phase. Future iterations should integrate vehicle delocalization directly into the optimization process to identify optimal routes and further reduce construction time.
- **Improved Optimization Process**: Following from above, we advise integrating the holistic objective function into the MIP optimizer in place of the simplified version, to better inform

decision-making. Additionally, parallelization could be explored to reduce computation time without the need for clustering.

By addressing these enhancements, the model can better account for real-world complexities, significantly improving its applicability and effectiveness across diverse project sites.

References

[1] International Organization for Standardization, "ISO 45001: Occupational health and safety," Accessed: Nov. 5, 2024. [Online]. Available: https://www.iso.org/iso-45001-occupational-health-and-safety.html

[2] California Department of Transportation, "Section 7-1 Environmental rules and requirements," Accessed: Nov. 5, 2024. [Online]. Available: https://dot.ca.gov/programs/construction/construction-manual/section-7-1-environmental-rules-and-requirements

[3] Climate Action Tracker, "USA," Accessed: Nov. 5, 2024. [Online]. Available: https://climateactiontracker.org/countries/usa/

[4] A. Mildenberger, "How renewable energy jobs can uplift fossil fuel communities and remake climate politics," Brookings, Feb. 28, 2023. [Online]. Available: https://www.brookings.edu/articles/how-renewable-energy-jobs-can-uplift-fossil-fuel-communities-and-remake-climate-politics

Appendix A: Meeting Notes with Charge Robotics

Nov 26th Meeting

- They liked the cost graph – can help them make decisions
    - Our cost is dependent on the time which can influence the graphs
- Adding to the cost:
    - Add in cost of deployment per factory or vehicle (cost of transporting to the site and back)
    - What's the estimate for this? Can get us a number by EOD
- Show them the report once we are finished
- Make sure to share graphs
- If you ever have questions we can email Isaac
- Meet next week so Max can be there – present the work we've done

Nov 19th Meeting

- Liked the tables – conveyed information well – liked the labels
    - Move it to the main body instead of appendix
- If he hadn't know how it works, it would've been confusing
    - We need more of a background, some diagrams
- Content seems great
- The more visuals the better
- The part with the labels it was hard to know what it was referring to until the end – more clear about that
- Main feedback:
    - Referring to table 2
    - We cannot have project go over the deadline – it is set and would hurt the companies reputation but they can throw money at the problem (ie get more factories, more vehicles)
    - This part is not emphasized in the report – we need to make sure to say that there is a **hard deadline** that has to be met
- Liked the scopes and the way that was organized

- Question about cost objective: would it be beneficial to minimize the number of factories to meet the deadline
    - They can use the algorithm to figure out how many factories they need
- We can use time as a proxy if it gets to complicated with cost

Basis for Comparison with the Data Set They Provided (how would they have built the site):

- Send them an empty site for them to show how they would've done it

- Steven might have some more data sets to send to us – can get it to us by tomorrow evening

Nov 12th Meeting

- How much time would it save in the end if the vehicles are delocalized
- Drivers are working like the uber app – they go to where they are needed
- Would be interesting to know total vehicles for optimization
- Can ask them for help wherever needed

- Send over the graphs to them
- If we need more testing data let them know

Nov 5th Meeting

- Reviewed scope with them and the focus on optimizing for time – feedback:
    - Wanted to see example of approaches so far
    - Mentioned scope division seems good and makes sense
        - 2A seems most natural next step then 3
    - Definitely consider factory movements as a priority
- Mixed integer program vs heuristic method (finding the nearest bays near factories)
- They can screen share future site with us (they have a new one) and run our code
- Have we considered roads?
    - Not yet
    - They mentioned might not be the most worthwhile thing

October 22nd Meeting

- Whats a bay (1 whole row or half the row, until the axis thing)
    - There are 7-10 bays per row
- Are 2 vehicles needed per bay or 1
- Go through assumptions we are making in our code
    - Assume all the bays are the same size
- We can ask them for technical help if needed as well
- Sanity check – 1 bay, building that with 1 factory – hand calculate and check
- Scope questions from their end:
    - Goal is to have a single factory no movement case by next week
    - Get a document with scope to them

October 15th Meeting

- Show them our example site and ask them how that normally gets built
    - Built rows nearest to the factory in the past
    - Might have different hardware on different parts of the site that would cause some restrictions
    - Just want to do it as fast as possible
- How big is the set of possible locations for the factory?
    - On the order of the 10s – wherever there are big clearings
- Row wise they are building inside out, but how does it happen for the column?
    - For each row, for each line, the middle is a column in the ground (it has a motor that rotates the entire thing) so they build from each side of the column
        - The motor – slew drive assembly
        - Can do it row by row, doesn't have to happen in the scale of the whole block
        - It's not a big gap, but a gap where it looks connected but there's no modules
        - Can build rows in whatever order that you want
        - Edge rows are built a bit differently to better withstand wind
    - Can assume that each row takes a fixed amount of time
- Assumptions for where the trucks can go:
    - They can go anywhere where its white space on our map
        - Would be nice if they didn't have to go between rows – have to drive super

carefully if its completed

Reviewing the Pseudocode:

- Shared with them on slack – will let us know if there's something drastically wrong

October 8th Meeting

- For sub-sites from the Mavericks site, will you provide us with sets of possible locations for mobile factories?
  - Look at places where there aren't solar builds
  - If there's a big inverter somewhere in the site we can assume that wasn't there
  - Assume we can't put factory it the middle of solar rows
  - In the future they would have an AutoCad with polygons on where the factories can go
- Review timeline/Report Feedback
  - Concerned that everything is linear – do you think it will take some iteration?
  - Very likely that model will change over time
  - Change contingency to sensitivity analysis for later steps in the report
  - More description for the appendix in the future for the final report (especially for equations)
  - Look into why it's not ctrl-F friendly
- Bay/Buffer
  - Bay can start building if there's something competed in the factory
  - If completed bay in factory is not cleared by the time bay is done, then it gets stuck → but also dependent on how big each bay is
- Mathematical model/optimization models
  - There is going to be interplay between the two – the optimization model you choose will impact the model
  - Matlab or python is good to use from their end

September 25th Meeting

- How to quantify output of where to put the factory (eg. gps coordinates, 1 factory per _ panels/block)
  - We will have places that the factory can be placed
- Validate the following:
  - 1 person per vehicle [4 delivery vehicles per factory]
    - Transit speed 24 km/hr [maximum]
    - 6 minute installation per bay once it gets to destination
  - 3 people per factory
    - 1 person monitoring software
    - 1 person in forklift loading pallets of solar modules & torque tubes
    - 1 person loading parts into the factory
  - 3 people on the field (mounting bays from delivery vehicle onto piles in the field)
  - 3 factories overall
  - 12 vehicles
- Do they value minimizing cost or time more?
  - Given a certain time of factores, what's the least amount of time we can complete this project → minimizing time allows for making for money and saving money → less people to pay for, more projects we can take on
  - Combination of using a reasonable number of factories and vehicles to complete the

project quickly

- Factory operating conditions/capacity?
  - 4-5 minutes per bay, 13-14 bays per hour [7 modules per bay]
  - Factory is pipelined – every stop in the factory lasts 38 seconds, an additional module is one more stop per bay
    - Removing a module from a bay saves time by 38 seconds – because it pipelined that's the base we can take
    - Can have longer bays at the end of of the rows and that would take a longer time
    - The first bay made might have a longer time because startup is needed – not really a big factor
  - Site is given in AutoCAD with where all the terrain is, where the panels will be going, so all of this is already considered – not something we need to worry about
- Do they work during the night?
  - 10 hour days starting 6am
    - Baseline hourly age [state dependent]: $25 USD to $40 USD
    - Overtime is 1.5x
  - Can't work nights, but sometimes work weekend if they are behind schedule
- Do they receive a site plan in advance? Is terrain considered in advance?
  - Yes
- Do these run in the rain?
  - Typically yes for rain
  - Wind speed impacts because of thinness of sheets: Maximum 20-24 km/hr
  - Don't worry about weather in optimization problem – just add for contingency
  - Add it into sensitivity analysis
- Can the trucks drive anywhere on the site?
  - They can't drive through arbitrary parts that don't have roads
  - Site plan will specify where the roads are
- During warmer conditions, is there any additional equipment needed?
  - Don't worry about it
- How long does it take to fix equipment? What are the costs associated with the repairs?
  - Average: 2 hours downtime [but they have no idea]
- Number of bays per truck – is it one?
  - One per vehicle
  - Buffer in the factory can only hold one bay, so the delivery vehicles needs to pick it up as another one is building
    - Buffer loads it into the delivery vehicle so its needed
  - 1 bay in buffer, 1 bay waiting [2 bays at a time basically]
- Where can we put the factories?
  - There are set places that they can be put
  - It is based on if the area is flat, access from roads, large enough clear area since the factory takes up some space, there are some places that is optimal to put the site
  - Mostly important to have flat areas
  - We can assume that we will be given a small list of where the factories can be put
  - Client + charge robotics figure this out together
- What are the regulatory constraints, and how are they quantified?
  - Labour – overtime will cost more
  - Regulations impact where the sites is layout but we don't have to worry about that
  - Regulations set the speed limits for the site [where the 24 km/hr came from]
  - Environmental regulations on where you can clear – all done before

- Time to move the factory?
    - Takes around 1 day [have to break down scaffolding, get a truck to move it]
    - They want to learn if its worth moving the factories from our analysis
    - For a large site it would make sense to move the factory, for a smaller site not maybe
    - You also have limited places to set up the factories
- Mavericks site is an ideal case – in the dessert and its pretty flat
- What other variable costs should we consider?
    - Gas/fuel
        - Not too high ~30kW generator
        - Factory draws a less than 10 kW
        - Efficiency is not known
    - Labour
    - Logistics costs – paying truckers if we are moving the site [might not be relevant for model]
    - Deicing solution needed for super cold – but don't worry about this, it's very rare
- Each factory can last 100k hours, costs around $1 million
    - Can assume we start at t = 0
- Each delivery vehicle costs around $150k – 75k hours
- What is the value of an extra day saved and an extra day late?
    - We can't be late – its a brick wall
    - Factor in a probability that each day/hour could have bad weather, or the factory will have a delay
    - If we save time, its saving depreciation on the factory + variable costs reduced [no bonus from the client] – reputation of finishing projects early means more projects
- Vehicle congestion?
    - Want parallelism – each vehicle assigned to a separate row
    - Model congestion
    - Unloading bays from the factory has some congestion too – 1 at a time
- Might get 1+ valid solutions – would be nice to see a few different candidate solutions and mention how they're different (one might have more factories but takes, one has less and takes shorter)

September 18th Meeting with Charge Robotics

Questions

- In the absence of optimization, how is the deployment of mobile factories currently planned? Are there a set of guiding principles? What are the biggest pain points or inefficiencies?
    - One project so far: small area - not much choice
    - Factory right next to area
    - Haven't received all layout details for next site
    - Figure out central-looking location - heuristic for what seems like a good spot
        - Will need it for bigger sites
- What are the key performance indicators (KPIs) that you use to evaluate the success of a deployment?
- Overall install time, cost of deployment (may conflict with time)
    - Three people associated with each factory, one with delivery vehicle, three in field - want to minimize
    - Time might be treated as constraint
    - Optimize overall install time [need to finish this project by a specific timeline, set by customers, constraint] – financial incentive to meet these timelines

        - Cost of deployment – # of people on the site [1 per vehicle, 3 on site for installation, want to minimize the number of people]
        - Sensitivity of system to mistakes e.g. what if a vehicle goes down, how does that impact install time
            - 12 delivery vehicles, 3 factories - not constraints, but might end up with them due to financial constraints
                - Cost of deployment not related to building vehicle, but should have variable for depreciation associated per day (e.g. lose $200 / day) - operational cost
                - Too many vehicles - congestion
            - Future scaled up company
            - Constraints for order in which sites get built
            - Between rows there's a gap connecting two halves: row needs to be built outwards from the center, doesn't constrain where factory needs to be placed
                - We can ignore this constraint - factor of solar tractor
            - Orientation of row is always from N-S, so panels turn from E-W
                - Single-axis tracts
- How is a route currently defined? Is the same mobile factory deployed for clustered sites?
    - Only done small sites for now
- What are the regulatory constraints, and how are they quantified? → Ask this in next meeting
- How is terrain and weather presently factored into deployment and plans?
    - Basically answered this
- What would a minimum viable product look like for you?
    - Answered below
- Could you please provide us with case studies for existing sites and what their deployment plans looked like? Do you collect any data during deployment that we can utilize?
    - Answered below
- What is your long-term vision for automating solar farm construction? Are you planning to expand this system to other types of infrastructure projects?
    - Moving towards larger sites on order of a few 100 MW
- How often would you like to meet, and what is the level of independence/collaboration with you on this project?
    - Regular meetings, communication via slack
- What is your timeline of deliverables on our end? What does the final deliverable look like
    - Report, source code for models
    - Recommendation

Isaac, Max, Steven

- Building robots that build solar farms: portable robotic factory deployed onto construction site
    - Power solar modules, brackets, fasteners
    - Robot arm pieces systems together into 'bays': 7-8 solar modules
    - At factory, loaded on delivery vehicles → field where workers construct
    - Some manual labour required for putting together bays
- Tubes dispensed into factory, built up
- Building massive solar site with two factories - how do we schedule the construction, where do we put the factories and move them around
- Multifaceted problem that they haven't truly figured out
- How many delivery vehicles?
- At scale: might be given contracts that require X amount of solar bays within 6 months, plan # factories, # vehicles, workers, placement of factories

- Site has features that will influence where factories should be e.g. roads, restricting or enabling ability to access sites
    - Out of scope? Planning for weather / seasons
    - Main variables: # vehicles, # bays and where they go, terrain / weather, # factories and their locations (initial placement, movement), rows (each row has 16 bays)
    - 100-500W modules
    - Each row had 100 modules, each module 500W → 50kW
    - Might be 4000 rows on a site → 200MW
    - Scale: 5 acres per MW
    - 1000 acres → 5 hectares
- Moving factories - time-cost of 1-2 days
- Not allowed to put factories in all locations
- Deliverables: report of how we're formulating problem mathematically, research on techniques, programs + code
- How often we're meeting them, as we need input: contact Isaac whenever, meet on short notice
- Program estimating time a construction plan takes ← writing this would validate information
- Particular site for case study: site plans are NDA, but plenty of solar sites on google maps - figure out layout
    - Steven will send us one specifically
- Want to send us close field data
- Hayworth California

How do they determine length of each row, where gaps are
- Based on terrain maybe

Do we have to account for number of brackets/fasteners per factory
Is the site layout known in advance?

Appendix B1: Code Excerpt of Simplified Objective Function *run-factory*

```
def run_factory(fact, bays, vehicles, sort):
    t = 0
    buffer = []
    installed = []
    t_return = np.zeros(vehicles)
    t_built = []
    avail_cars = []

    if sort == True:
        bays = sort_by_dist(bays, fact)
    planned = list(bays)

    while installed != bays:
        for i in range(vehicles):
            # car has returned from delivery + install
            if t == t_return[i]:
                avail_cars.append(i)

        # building a new bay
        if len(buffer) < 2 and len(planned) > 0:
            if len(buffer) == 1:
                if t >= t_built[0]:
                    build_bay = planned.pop(0)
                    t_built.append(build(build_bay, t))
                    buffer.append(build_bay)
            else:
                build_bay = planned.pop(0)
                t_built.append(build(build_bay, t))
                buffer.append(build_bay)

        # delivering and installing a built bay
        if len(avail_cars) > 0 and len(buffer) > 0:
            if t >= t_built[0]:
                t_built.pop(0)
                deliver_bay = buffer.pop(0)
                installed.append(deliver_bay)
                car = avail_cars.pop(0)
                t_return[car] = deliver(fact, deliver_bay, t)

        t += 1
    t_finish = np.max(t_return)
    return t_finish

def build(bay, t):
    T_bay = 5 # 4-5 min to assembly
    t_built = t + T_bay
    return t_built
```

```
def deliver(fact, bay, t):
    v = 400 # 24 km/h = 400 m/min
    T_install = 6 # 6 min to install

    distance = dist(fact, bay)
    travel = 2 * np.ceil(distance / v)
    #print(travel)
    t_return = t + travel + T_install
    return t_return

def dist(fact, bay):
    x_dist = np.abs(fact[0] - bay[0])
    y_dist = np.abs(fact[1] - bay[1])
    return x_dist + y_dist
```

Code Excerpt 1. Simplified sequential objective function ***run-factory***

Appendix B2: Code Excerpt of Holistic Objective Function *run-factories*

```
def run_factories(factory_assignments, vehicles):
    # facts: list of factory locations, used to index fact_bay
    # fact_bay: dictionary of bays assigned to factories
    t = 0

    # site trackers
    n_factories = len(list(factory_assignments.keys()))
    loc_list = list(factory_assignments[0].keys()) # all factory locations
    n_locations = len(loc_list)
    planned = copy.deepcopy(factory_assignments)
    installed = []
    bays = []
    for j in range(n_factories):
        for fact, assigned_bays in planned[j].items():
            for bay in assigned_bays:
                bays.append(bay)

    # factory trackers
    buffer = {j: [] for j in range(n_factories)}
    t_built = {j: [] for j in range(n_factories)}
    t_moved = np.zeros(n_factories) # keep track of move-finish times for
                    each factory
    fact_index = np.zeros(n_factories, dtype=int) # factory's current
                    location index
    cur_fact = [(0,0)]*n_factories # factory's current location
    fact_complete = np.zeros(n_factories) # = 1 when no more factory
                    locations for factory j
    factory_start_time = {j: [] for j in range(n_factories)} # = -1 if no
                    bays assigned
    factory_finish_time = {j: [] for j in range(n_factories)}

    #vehicle trackers
    t_delivered = np.zeros(vehicles) # time each vehicle finishes delivery
    avail_cars = []
    car_loc = [(0,0)]*vehicles

    # initializing first location for each factory
    for j in range(n_factories):
        for l in range(n_locations):
            fact = loc_list[l]
            if len(planned[j][fact]) > 0:
                fact_index[j] = l
                cur_fact[j] = fact
                factory_start_time[j].append(0)
                break
            else:
                factory_start_time[j].append(-1)
```

```
            factory_finish_time[j].append(-1)

    while len(installed) < len(bays):
        for i in range(vehicles):
            # car has finished delivery + install
            if t == t_delivered[i]:
                avail_cars.append(i)

      # resorting facts so that no single fact hogs the newly available car
        remaining = [len(planned[j][cur_fact[j]]) for j in
                    range(n_factories)]
        j_sorted = [j for _, j in sorted(zip(remaining, [j for j in
                    range(n_factories)]), reverse=True)]

        for j in j_sorted:
            if len(buffer[j]) < 2 and len(planned[j][cur_fact[j]]) > 0 and
                    t >= t_moved[j]:
                # factory has finished moving, can build a bay
                if (len(buffer[j]) == 1 and t >= t_built[j][0]) or
                    len(buffer[j]) == 0:
                    # previous bay assembled or buffer is empty
                    build_bay = planned[j][cur_fact[j]].pop(0)
                    t_built[j].append(build(t))
                    buffer[j].append(build_bay)

            elif len(planned[j][cur_fact[j]]) == 0 and fact_complete[j] ==
                0 and len(buffer[j]) == 0:
                # for this factory, all bays for location are installed
                prev_fact = cur_fact[j]
                factory_finish_time[j].append(t)
                # only check factories after current factory
                for l in range(fact_index[j]+1, n_locations):
                    if len(planned[j][loc_list[l]]) > 0: # if next location
                                        has assigned bays
                        fact_index[j] = l
                        cur_fact[j] = loc_list[l] # set new factory
                                        location for factory j
                        break
                    else:
                        factory_start_time[j].append(-1)
                        factory_finish_time[j].append(-1)
                if prev_fact == cur_fact[j]:
                    # signifies that there are no more factory locations
                                        for factory j
                    fact_complete[j] = 1
                else:
                    factory_start_time[j].append(t)
                    # loc start is when the factory is relocated there
                    t_moved[j] = move(t)
```

```
            if len(avail_cars) > 0 and len(buffer[j]) > 0:
                if t >= t_built[j][0]:
                    # if there are available cars and bays to be relieved
                    t_built[j].pop(0)
                    deliver_bay = buffer[j].pop(0)
                    installed.append(deliver_bay)

                    car_min_dist = np.inf
                    car_min = 0
                    # sorting available cars by distance to factory
                    for c in range(len(avail_cars)): # avail_cars: 2, 0, 1
                        car_dist = dist(car_loc[avail_cars[c]], fact)
                        if car_dist < car_min_dist:
                            car_min_dist = car_dist
                            car_min = c
                    car = avail_cars.pop(car_min)
                    fact = cur_fact[j]
                    t_delivered[car], car_next = deliver(fact, deliver_bay,
                                          t, car_loc[car])
                    car_loc[car] = car_next

        t += 1

    t_finish = np.max(t_delivered)
    cost = estimate_cost(t_finish, vehicles, n_factories)
    print("Finish Building at t = %i min" %t_finish)
    print("Cost estimate: %i USD" %cost)
    return t_finish, factory_start_time, factory_finish_time, cost

def build(t):
    T_bay = 5 # 4-5 min to assembly
    t_built = t + T_bay
    return t_built

def deliver(fact, bay, t, car_cur):
    v = 400 # 24 km/h = 400 m/min
    T_install = 6 # 6 min to install

    dist_retrieve = dist(car_cur, fact)
    dist_deliver = dist(fact, bay)
    car_next = bay # car's next location is at delivered bay
    t_travel = np.ceil((dist_retrieve + dist_deliver) / v)
    t_delivered = t + t_travel + T_install
    return t_delivered, car_next

def move(t):
    T_move = 60 * 10 # 10-hour work day
    t_moved = t + T_move
```

```
    return t_moved

def dist(fact, bay):
    x_dist = np.abs(fact[0] - bay[0])
    y_dist = np.abs(fact[1] - bay[1])
    return x_dist + y_dist
```

*Code Excerpt 2.1. Holistic sequential objective function **run-factories***

```
factory_assignments = {0: {(0,0): [(100,100),(200,200)],
                           (10,10): [],
                           (20,20): [(300,300)],
                           (30,30): []},
                       1: {(0,0): [],
                           (10,10): [(150,150)],
                           (20,20): [],
                           (30,30): [(250,250)]}}
```

*Code Excerpt 2.2. Example of input to **run-factories***

Appendix B3: Code Excerpt of Cost Estimator Function

```
def estimate_cost(time, vehicles, factories):
    hours = time / 60

    # factory and vehicle depreciation rate
    d_factory = 10
    d_vehicle = 2

    # labor parameters
    c_labor = 25
    N_factory = 3
    N_field = 3
    N_vehicle = 1

    # Flat deployment cost per factory and per vehicle
    flat_cost_per_factory = 35000
    flat_cost_per_vehicle = 6000

    factory_cost = hours * factories * d_factory
    vehicle_cost = hours * vehicles * d_vehicle
    factory_labor = factories * (N_factory + N_field)
    vehicle_labor = vehicles * N_vehicle
    labor_cost = hours * c_labor * (factory_labor + vehicle_labor)

    flat_cost = factories * flat_cost_per_factory + vehicles *
flat_cost_per_vehicle

    total_cost = factory_cost + vehicle_cost + labor_cost + flat_cost
    return total_cost
```

Code Excerpt 3. Cost Estimator Function

Appendix B4: Code Excerpt of MIP Problem Formulation

```
def multi_MIP(bays, facts, n_vehicles, n_factories, num_clusters=50):
    # Cluster bay locations to reduce problem size
    cluster_centers, bay_weights, labels = cluster_bays(bays,
num_clusters=num_clusters)
    facts_L = facts
    cluster_centers = np.array(cluster_centers)
    facts = np.array(facts)
    n_clusters = len(cluster_centers)
    n_locations = len(facts)

    problem = pl.LpProblem("k_median_moving_factory", pl.LpMinimize)

    # === Decision Variables ===
    X = pl.LpVariable.dicts("service_bay", indices=(range(n_clusters),
range(n_factories), range(n_locations)), lowBound=0, upBound=1,
cat='Binary')
    Y = pl.LpVariable.dicts("move_factory", indices=(range(n_factories),
range(n_locations)), lowBound=0, upBound=1, cat='Binary')
    max_time = pl.LpVariable('max_time', lowBound=0, cat='Continuous')

    # === Objective Function ===
    movement_penalty = 60 * 10 # 10-hour work day
    problem += max_time  # Objective function to minimize

    # Add terms to the objective function, weighted by the number of bays
in each cluster
    for j in range(n_factories):
        problem += (sum(Y[j][k] for k in range(n_locations)) - 1) *
movement_penalty + \
                   sum(run_factory(facts[k], [cluster_centers[i]],
n_vehicles, True) * X[i][j][k] * bay_weights[i]
                       for k in range(n_locations) for i in
range(n_clusters)) <= max_time

    # === Constraints ===
    # Each cluster is serviced by exactly one factory
    for i in range(n_clusters):
        problem += sum(X[i][j][k] for k in range(n_locations) for j in
range(n_factories)) == 1

    # Each location can be served by at most one factory
    for k in range(n_locations):
        problem += sum(Y[j][k] for j in range(n_factories)) <= 1

    # A factory can only service a bay cluster if it's active at the
location
    for i in range(n_clusters):
```

```
        for j in range(n_factories):
            for k in range(n_locations):
                problem += X[i][j][k] <= Y[j][k]

    # === Solve the problem ===
    problem.solve(solver=PULP_CBC_CMD(msg=True, timeLimit=120*5))
```

Code Excerpt 4. MIP problem formulation

Appendix B6: Code Excerpt of K-Means Cluster Creation

```
def cluster_bays(bays, num_clusters=1000):
    # Convert bay locations to a numpy array if they aren’t already
    bays_array = np.array(bays)

    # Perform clustering
    kmeans = KMeans(n_clusters=num_clusters,
random_state=0).fit(bays_array)

    # Get cluster centroids, which will serve as representative locations
for grouped bays
    cluster_centers = kmeans.cluster_centers_
    # Count the number of bays in each cluster
    bay_weights = np.bincount(kmeans.labels_)

    return cluster_centers, bay_weights, kmeans.labels_
```

Code Excerpt 5. Cluster creation

Appendix C: Hand Calculations

Excel sheet linked here.

Factory 1: (0,0)			
t=1			
t=2			
t=3			
t=4			
t=5	built	build new	
t=6			
t=7	delivered		
t=8			
t=9			
t=10		built - in buffer	
t=11			
t=12			
t=13	finish install		
t=14			
t=15	get back	grab new	build new
t=16			
t=17	delivered		
t=18			
t=19			
t=20			built - in buffer
t=21			
t=22			
t=23	finish install		
t=24			
t=25	get back	grab new	build new
t=26			
t=27	delivered		
t=28			
t=29			
t=30			
t=31			
t=32			
t=33	finish install		
t=34			
t=35	get back		

Factory 2: (200,200)			
t=1			
t=2			
t=3			
t=4			
t=5	built	build new	
t=6	delivered		
t=7			
t=8			
t=9			
t=10		built - in buffer	
t=11			
t=12	finish install	grab new	build new
t=13	get back		
t=14	delivered		
t=15			
t=16			
t=17			built - in buffer
t=18			
t=19			
t=20	finish install		
t=21	get back	grab new	build new
t=22	delivered		
t=23			
t=24			
t=25			
t=26			
t=27			
t=28	finish install		
t=29	get back		

Factory 3: (100,400)			
t=1			
t=2			
t=3			
t=4			
t=5	built	build new	
t=6			
t=7	delivered		
t=8			
t=9			
t=10		built - in buffer	
t=11			
t=12			
t=13	finish install		
t=14			
t=15	get back	grab new	build new
t=16			
t=17	delivered		
t=18			
t=19			
t=20			built - in buffer
t=21			
t=22			
t=23	finish install		
t=24			
t=25	get back	grab new	build new
t=26			
t=27	delivered		
t=28			
t=29			
t=30			
t=31			
t=32			
t=33	finish install		
t=34			
t=35	get back		

Case 1. Hand Calculations for building 3 panels with 1 vehicle

Factory 1: (0,0)				
t=1				
t=2				
t=3				
t=4				
t=5	built	build new		
t=6				
t=7	V1 delivered			
t=8				
t=9				
t=10		built - in buffer	build new	
t=11				
t=12				V2 delivered
t=13	V1 installed			
t=14				
t=15	V1 get back		built - in buffer	
t=16				
t=17	V1 delivered			V2 installed
t=18				
t=19				
t=20				
t=21				
t=22				
t=23	V1 installed			
t=24				
t=25	get back			

Factory 2: (200,200)				
t=1				
t=2				
t=3				
t=4				
t=5	built	build new		
t=6	V1 delivered			
t=7				
t=8				
t=9				
t=10		built - in buffer	build new	
t=11				
t=12	v1 installed			
t=13	V1 back			
t=14				
t=15			built	
t=16	V1 delivered			
t=17				V2 installed
t=18				
t=19				
t=20				
t=21				
t=22	v1 installed			
t=23	v1 back			

Factory 3: (100,400)				
t=1				
t=2				
t=3				
t=4				
t=5	built	build new		
t=6				
t=7	V1 delivered			
t=8				
t=9				
t=10		built - in buffer	build new	
t=11				
t=12				V2 delivered
t=13	V1 installed			
t=14				
t=15	V1 back	grab new	built - in buffer	
t=16				
t=17	V1 delivered			
t=18				V2 installed
t=19				
t=20			built - in buffer	
t=21				
t=22				
t=23	V1 installed			
t=24				
t=25	V1 back			

Case 2. Hand Calculations for building 3 panels with 2 vehicles

Factory 1: (0,0)			
t=1			
t=2			
t=3			
t=4			
t=5	built (1)	V1 picks up	build new
t=6			
t=7		V1 delivers	
t=8			
t=9			
t=10			built (2)
t=11			
t=12			
t=13		V1 done	
t=14			
t=15	build new	V1 back, pick up (2)	
t=16			
t=17		V1 delievers	
t=18			
t=19			
t=20	built (3)		
t=21			
t=22			
t=23		V1 done	
t=24			
t=25	build new	V1 back, pick up (3)	
t=26			
t=27		V1 delievers	
t=28			
t=29			
t=30	built (4)		
t=31			
t=32			
t=33		V1 done	
t=34			
t=35	new build	V1 back, pick up 4	
t=36			
t=37		V1 delievers	
t=38			
t=39			
t=40	built (5)		
t=41			
t=42			
t=43		V1 done	
t=44			
t=45	new build	V1 back, pick up 5	
t=46			
t=47		V1 delievers	
t=48			
t=49			
t=50			
t=51	built (6)		
t=52			
t=53		V1 done	
t=54			
t=55		V1 back, pick up 6	
t=56			
t=57			
t=58		V1 delievers	
t=59			
t=60			
t=61			
t=62			
t=63			
t=64		V1 done	
t=65			
t=66			
t=67		V1 back	

Factory 1: (200, 200)			
t=1			
t=2			
t=3			
t=4			
t=5	built (1)	V1 picks up	build new
t=6		V1 delivers	
t=7			
t=8			
t=9			
t=10			built (2)
t=11			
t=12		V1 done	
t=13	build new	V1 back, pick up (2)	
t=14		V1 delievers	
t=15			
t=16			
t=17			
t=18	built (3)		
t=19			
t=20		V1 done	
t=21	build new	V1 back, pick up (3)	
t=22		V1 delievers	
t=23			
t=24			
t=25			
t=26	built (4)		
t=27			
t=28		V1 done	
t=29	new build	V1 back, pick up 4	
t=30			
t=31		V1 delievers	
t=32			
t=33			
t=34	built (5)		
t=35			
t=36			
t=37		V1 done	
t=38			
t=39	new build	V1 back, pick up 5	
t=40		V1 delievers	
t=41			
t=42			
t=43			
t=44	built (6)		
t=45			
t=46		V1 done	
t=47		V1 back, pick up 6	
t=48			
t=49		V1 delievers	
t=50			
t=51			
t=52			
t=53			
t=54			
t=55		V1 done	
t=56			
t=57		V1 back	

Factory 1: (100, 400)			
t=1			
t=2			
t=3			
t=4			
t=5	built (1)	V1 picks up	build new
t=6			
t=7		V1 delivers	
t=8			
t=9			
t=10			built (2)
t=11			
t=12			
t=13		V1 done	
t=14			
t=15	build new	V1 back, pick up (2)	
t=16			
t=17		V1 delievers	
t=18			
t=19			
t=20	built (3)		
t=21			
t=22			
t=23		V1 done	
t=24			
t=25	build new	V1 back, pick up (3)	
t=26			
t=27		V1 delievers	
t=28			
t=29			
t=30	built (4)		
t=31			
t=32			
t=33		V1 done	
t=34			
t=35	new build	V1 back, pick up 4	
t=36			
t=37		V1 delievers	
t=38			
t=39			
t=40	built (5)		
t=41			
t=42			
t=43		V1 done	
t=44			
t=45	new build	V1 back, pick up 5	
t=46			
t=47		V1 delievers	
t=48			
t=49			
t=50			
t=51	built (6)		
t=52			
t=53		V1 done	
t=54			
t=55		V1 back, pick up 6	
t=56			
t=57		V1 delievers	
t=58			
t=59			
t=60			
t=61			
t=62			
t=63		V1 done	
t=64			
t=65		V1 back	

Case 3. Hand Calculations for building 6 panels with 1 vehicle

Factory 1: (0,0)				
t=1				
t=2				
t=3				
t=4				
t=5	built (1)	V1 picks up	build new	
t=6				
t=7		V1 delivers		
t=8				
t=9				
t=10	build new		built (2)	V2 picks up
t=11				
t=12				V2 delievers
t=13		V1 done		
t=14				
t=15	built (3)	V1 back	build new	
t=16				
t=17		V1 delievers		
t=18				V2 done
t=19				
t=20	build new		built (4)	V2 back
t=21				
t=22				V2 delievers
t=23				
t=24		V1 back		
t=25	built (5)	V1 picks up	new build	
t=26				
t=27		V1 delievers		
t=28				V2 done
t=29				
t=30			built (6)	V2 back
t=31				
t=32				
t=33		V1 done		V2 delievers
t=34				
t=35		V1 back		
t=36				
t=37				
t=38				
t=39				V2 done
t=40				
t=41				
t=42				V2 back

Factory 2: (200,200)				
t=1				
t=2				
t=3				
t=4				
t=5	built (1)	V1 picks up	build new	
t=6		V1 delivers		
t=7				
t=8				
t=9				
t=10	build new		built (2)	V2 picks up
t=11				V2 delievers
t=12		V1 done		
t=13		V1 back		
t=14				
t=15	built (3)	V1 picks up	build new	
t=16		V1 delievers		
t=17				V2 done
t=18				V2 back
t=19				
t=20	build new		built (4)	V2 picks up
t=21				
t=22		V1 back		V2 delievers
t=23				
t=24				
t=25	built (5)	V1 picks up	new build	
t=26		V1 delievers		
t=27				
t=28				V2 done
t=29				
t=30			built (6)	V2 back
t=31				
t=32		V1 done		V2 delievers
t=33		V1 back		
t=34				
t=35				
t=36				
t=37				
t=38				V2 done
t=39				
t=40				V2 back

Factory 3: (100,400)				
t=1				
t=2				
t=3				
t=4				
t=5	built (1)	V1 picks up	build new	
t=6				
t=7		V1 delivers		
t=8				
t=9				
t=10	build new		built (2)	V2 picks up
t=11				
t=12				V2 delievers
t=13		V1 done		
t=14				
t=15	built (3)	V1 back	build new	
t=16				
t=17		V1 delievers		
t=18				V2 done
t=19				
t=20	build new		built (4)	V2 back
t=21				
t=22				V2 delievers
t=23				
t=24		V1 back		
t=25	built (5)	V1 picks up	new build	
t=26				
t=27		V1 delievers		
t=28				V2 done
t=29				
t=30			built (6)	V2 back
t=31				
t=32				V2 delievers
t=33		V1 done		
t=34				
t=35		V1 back		
t=36				
t=37				
t=38				V2 done
t=39				
t=40				V2 back

Case 4. Hand Calculations for building 6 panels with 2 vehicles

Appendix D: Test Results for Time-Cost Investigation

# Factories	*# Vehicles*	Project Time (Days)	Cost (USD $)
1	1	109.4	531,99.67
1	2	71.93	416,449.60
1	3	71.88	468,781.23
1	4	71.88	521,362.53
1	5	71.88	573,943.83
1	6	71.88	626,525.13
1	7	71.88	679,106.43
1	8	71.88	731,687.73
1	9	71.88	784,269.03
2	1	134.39	1,195,179.10
2	2	64.58	661,656.37
2	3	42.64	498,363.35
2	4	35.85	462,201.27
2	5	35.81	491,072.50
2	6	35.81	520,279.00
2	7	35.81	549.485.50
2	8	35.81	578,692.00
2	9	35.81	607,898.50
3	1	142.65	1,725,144.35
3	2	67.09	976,873.50
3	3	44.74	725,411.15
3	4	34.40	614,511.60
3	5	26.35	523,915.75
3	6	24.00	510,781.30
3	7	24.00	532,277.10

3	8	24.00	553,826.40
3	9	24.00	575,375.70