Scalable and Conflict Free Routing Algorithms for Large Fleets of Mobile Robots

Master's thesis in System, Control and Mechatronics

MARCUS DEGERMAN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Scalable and Conflict Free Routing Algorithms for Large Fleets of Mobile Robots

MARCUS DEGERMAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Scalable and Adaptive Routing Algorithms for Large Fleet Management
MARCUS DEGERMAN

Advisor: Philip Rasko, Kollmorgen Automation AB
Supervisor: Doctor Sabino Roselli, Department of Electrical Engineering
Examiner: Professor Martin Fabian, Department of Electrical Engineering

Master's Thesis 2023
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

MARCUS DEGERMAN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis explores methods for pathfinding and scheduling for large fleets of mobile robots, such as those used for material handling in factories and warehouses. The intention is to be able to handle larger fleets than is currently possible by conventional means.

A path is a sequence of segments in a directed graph representing the warehouse layout in which the robots move, and pathfinding determines the paths from starting points to destinations. Scheduling then determines at which time each section of a path is travelled. For the robots not to deadlock each other, a schedule has to be conflict free, and we also want the schedule to minimise the overall work time.

As conflicts can only occur when paths overlap, it can be hypothesized that choosing to schedule paths that minimise overlaps could potentially lessen the computational burden of scheduling, as well as minimising overall work time. The thesis demonstrates that identifying paths that minimise overlaps has this effect, but is itself a computationally challenging task.

Keywords: Autonomous Guided Vehicles (AGV), Vehicle Routing Problem (VRP), Optimization, Satisfiability Modulo Theories (SMT), Mixed Integer Linear Programming (MILP)

Acknowledgements

This master's thesis has been conducted at Kollmorgen Automation in Gothenburg, Sweden, in conjunction with Chalmers University of Technology, Department of Electrical Engineering in Gothenburg, Sweden, during the spring of 2023.
My work would not have been possible without the tremendous help and assistance from the people in the stationary team at Kollmorgen. During my time with the Kollmorgen stationary team, I received inputs, tips, guidance and maybe, most importantly, a very warm welcome. I would like to thank Martin Törnqvist, the manager of the stationary team, for the equipment to complete the thesis work and the whole team for a terrific work environment.
I want to give a special thanks to my supervisor at Kollmorgen, Philip Rasko, but also to Fredrik Hagebring. Their patience, valuable insight and ideas have been of great value and pointed me in the right direction throughout the master thesis work. Additionally, I am deeply grateful to my supervisor at Chalmers, Sabino Roselli and my examiner, Martin Fabian, for their guidance in crafting a compelling report and for making this project possible in the first place.

Marcus Degerman, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AGV	Autonomous Guided Vehicles
AS/RS	Automated Storage and Retrieval Systems
CBS	Conflict Based Search
CF-EVRP	Conflict-Free Electric Vehicle Routing Problem
CNF	Conjunctive Normal Form
CVRP	Capacitated Vehicle Routing Problem
IP	Integer Programming
JSP	Job Shop Problem
LP	Linear Programming
MAPF	Multi Agent Path Finding
MILP	Mixed Integer Linear Programming
OMS	Order Management System
SAT	Boolean Satisfiability
SMT	Satisfiability Modulo Theories
SP	Shortest Path
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
W_SP	Weighted Shortest Path

Contents

List of Figures

List of Tables

1

Introduction

As industries are increasingly automated, transporting goods inside the plant must be more flexible. This can be done using Autonomous Guided Vehicles (AGVs) to transport goods to the right place at the right time [1]. Kollmorgen Automation AB operates in material handling through automated vehicles and is interested in computing the paths and schedules of these AGVs. A path is a sequence of road segments in the plant the vehicles should travel through [2], while a schedule assigns the time when an AGV traverse a specific segment of the path it is assigned [3].

A schedule for a few AGVs or a small area does not pose significant challenges [2]. However, computational complexity grows when more AGVs are present in the manufacturing facility or the operating space expands, resulting in a complex scheduling process that is computationally demanding.

The complexity of scheduling [3] AGVs arises from two primary factors: the number of AGVs that must be scheduled and the number of conflicts between the AGVs. Conflicts can arise when paths overlap, i.e., when two or more vehicles are assigned paths that share the same nodes or are near each other. The more vehicles there are, the more chances there are for conflicts.

In situations where there is no overlap, scheduling becomes straightforward. Under these conditions, each AGV can operate independently without coordination. The lengthiest task would determine the overall completion time or makespan. On the other hand, if there is complete overlap, only one AGV can function at a given time, simplifying the scheduling process. In this case, the makespan is equivalent to the sum of the time durations of all tasks. Therefore, scheduling becomes trivial at both extremes of no overlap and complete overlap.

One way to schedule the AGVs is to model the problem as a Satisfiability Modulo Theory (SMT) problem and solve it using an SMT solver, as described in [4]. Some SMT solvers can perform optimisation, i.e. it is possible to define an objective function of a subset of the problem variables and maximise/minimise it. The function to maximise/minimise can include overlaps between paths, conflicts between vehicles, the distance travelled, or battery used, but many others. For the specific problem tackled in this project, overlaps between paths will be minimised.

It has been reported in previous research [5] that an SMT solver can find a good enough solution regarding the cost function in a relatively short period, whereas identifying the optimal solution and confirming it is the optimal solution requires significantly more time.

1.1 Research objective

As previously outlined, the scheduling process becomes considerably less complicated in cases of no overlap or total overlap. However, such extremes are seldom the reality. It could be assumed that selecting paths that minimise overlap might yield better schedules, decreased computation time and a lower makespan than a standard scenario. The natural standard scenario or base case uses shortest paths.

Therefore, this thesis explores whether general-purpose solvers can contribute to reducing the computation time and the resulting makespan required for scheduling by computing paths that minimises overlaps, compared to using shortest paths. The aim is to empirically assess this strategy across various scenarios, focusing on its performance in large-scale systems.

As a result, the research question formulated for this study is:

- Can selecting paths with minimal overlaps shorten the computation time for scheduling, and the resulting makespan, compared to using the shortest paths?

1.2 Limitations

Since this thesis focuses on evaluating whether paths that are computed to minimise the number of overlaps are faster to schedule than using shortest paths, a few limitations are put in place to narrow the scope of the thesis.

The process of assigning each vehicle's destination, as well as their starting positions, would, in the real system, be computed by an external system called *Order Manager* (see Section 3.4 for further information about it), but during the thesis they will be randomised. This will be done in such a way that the all pairs can be retested.

Furthermore, all vehicles are assumed to be able to reach their intended goals. This assumption is validated in two ways. Firstly, the goals do not overlap, allowing each vehicle to reach its destination without hindrance. If a vehicle still hinders another vehicle from reaching its goal, they are assumed to move out of the way, clearing the path for the other vehicle. This assumption ensures the smooth operation of the fleet and, by extension, the validity of the scheduling process being evaluated.

While the scheduling problem in question could be modelled using Mixed Integer Linear Programming (MILP), research [5] suggests that Satisfiability Modulo Theories (SMT) are more efficient in this area. As a result, MILP will not be used for that part. Instead, it will be used to create the paths according to the minimisation of the number of overlaps, as after a few small experiments, it was found to be much faster than SMT for this purpose.

It is not the most optimal path or solution sought after but a solution that is good enough and is found or calculated in a reasonable time.

1.3 Related Work

Dijkstra's algorithm [6], is a well-established method for finding the shortest path between two nodes in a graph. In the context of this thesis, Dijkstra's algorithm is used for finding paths for one vehicle at a time, independent of others.

A* [7] is another pathfinding algorithm which uses a heuristic to guide its search. However, finding an appropriate heuristic can be challenging. Experimental comparison in the given context revealed that the time to find paths using Dijkstra's algorithm was negligible compared to using the optimisation software Gurobi. Therefore, there was no need to attempt to speed up Dijkstra's algorithm by replacing it with A*.

Multi-agent pathfinding (MAPF) [8] addresses the challenge of devising non-intersecting paths for multiple vehicles. This seminal work, highlighting the intricacies of coordinated movement in a shared environment, served as an essential reference at the outset of the thesis.

Scheduling [3] is another vital component of the system. In general, scheduling is arranging, controlling and optimising work and workloads in a production or manufacturing process. In this thesis, the concept of scheduling is used and expanded upon to consider additional factors and constraints.

The Job-Shop Scheduling Problem (JSP) [2] is a classic problem in operations research and computer science. It involves scheduling a set of jobs in a machine shop, where each job consists of tasks that need to be completed in a specific order, and each task requires processing on a specific machine for a fixed length of time. In the context of this thesis, the Job-Shop Scheduling Problem serves as a foundation but is further expanded.

The Vehicle Routing Problem (VRP) [9] is a classic problem in logistics and operations research. However, it is not necessarily the classical problem addressed in the thesis. The VRP traditionally involves determining optimal routes for vehicles to deliver goods to customers to minimise the total distance travelled or cost but does not consider conflicts between that vehicles.

The Conflict-Free Electric Vehicle Routing Problem (CF-EVRP) [10] is a combinatorial optimisation problem of designing routes for vehicles to visit customers such that a cost function, typically the number of vehicles or the total travelled distance, is minimised. The CF-EVRP involves constraints such as time windows on the delivery to the customers, limited operating range of the vehicles, and limited capacity on the number of vehicles a road segment can accommodate simultaneously. Part of this work acted as a foundation for the scheduling but was modified to use different constraints.

Conflict-based search (CBS) [11] is an algorithm used for solving Multi-Agent Path Finding (MAPF) problems. CBS identifies and resolves conflicts among agents' paths to ensure collision-free navigation. It operates on two levels: the high level, where conflicts are detected, and the low level, where each conflict is resolved by creating a constraint for one of the agents involved in the conflict and generating a new path for that agent. CBS is noteworthy for its efficient performance and

ability to provide optimal solutions for MAPF problems, considering the total sum of agents' path lengths. It presents a high potential in various applications, including automated warehouses and autonomous vehicles. Conflict-based search could apply to the problems considered in the thesis. However, it is too different from the approach taken by Kollmorgen, and with the author's current understanding, it would also be challenging to implement.

1.4 Thesis Outline

This thesis is structured into four primary sections. Firstly, the Problem Definition (Section 2) outlines the specific challenges the thesis seeks to address, setting the scene by highlighting the existing gaps in knowledge and the importance of the problem. Next, the Problem Formulation (Section 3) delves into a detailed description of how the problem is structured for analysis and solution, including developing appropriate models and techniques. The third part, Evaluation (Section 4), presents the methodology and results of testing the proposed solutions. Lastly, the Conclusion (Section 5) provides a conclusion of the findings, exploring their implications, comparing them with existing knowledge, and suggesting future avenues for research.

2

Problem Definition

This chapter offers an overview of various concepts and issues central to this thesis. It begins with an introduction to Vehicle Routing Problems (VRPs), outlining their general nature and importance in modern transportation and logistics.

The discussion then focuses on Automated Guided Vehicles (AGVs), explaining their role and operational principles as integral components of contemporary warehousing and manufacturing facilities.

The chapter also explains the concept of General Purpose solvers. These computational tools are pivotal for tackling complex optimisation problems, such as the ones encountered in AGV scheduling. The exploration of these solvers provides an understanding of their strengths and limitations, preparing the ground for their application in this thesis.

Lastly, this chapter delves into some specific challenges this thesis addresses. These include the complexities of real-time computation, the difficulties posed by large-scale problems, and the intricacies of managing and negotiating complex constraints. These challenges represent a significant hurdle in efficiently implementing AGVs and VRPs. Thus, this chapter provides an understanding of these hurdles.

2.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a well-known optimisation problem in operations research and logistics of finding the most efficient way to deliver goods or services to customers using a fleet of vehicles. The problem can be stated as follows: given a set of customers with known demands and locations, a set of vehicles with limited capacity, and a depot from which the vehicles start and return, the goal is to determine the optimal set of routes for the vehicles to serve all the customers while minimising the total distance travelled or the total cost of transportation.

The VRP is a complex problem that can take many forms depending on the specific constraints and objectives of the application. Some common variations of the VRP include the Capacitated Vehicle Routing Problem (CVRP) [12], which assumes that each vehicle has a maximum capacity, and the Vehicle Routing Problem with Time Windows (VRPTW) [13], which adds the constraint that each customer can only be served within a specific time.

Solving the VRP to optimality can be challenging due to the ample search space and

the many routes the vehicles can take. Many different algorithms and techniques can be used to solve the VRP, including exact methods such as branch and bound [9], heuristic methods such as tabu search [14] or simulated annealing, and metaheuristic methods such as genetic algorithms [15] or ant colony optimisation [16].

The problem tackled in this work is not strictly a classical vehicle routing problem. It is more appropriately characterised as planning and scheduling paths for a fleet of mobile robots, such as AGVs. This problem is often called Multi-agent pathfinding (MAPF) [8]. This process involves computing paths and managing vehicle interactions to prevent collisions or conflicts.

To accurately represent the operational environment and facilitate efficient path planning and execution, the facility's layout is modelled as a directed weighted graph. Importantly, this study utilises real-world layouts provided by Kollmorgen. These layouts include information about where overlaps may occur and, thus, potential conflicts.

2.2 Automated Guided Vehicles

Automated Guided Vehicles (AGVs) are robotic vehicles designed to transport materials, goods or products in manufacturing plants, warehouses, distribution centres, and other industrial or commercial facilities [1]. AGVs are typically equipped with sensors, cameras, and other navigational equipment that allow them to move autonomously and safely through their environment without human intervention. AGVs can also be integrated with other systems, such as automated storage and retrieval systems (AS/RS), conveyor systems, or robotic work cells, to create a fully automated manufacturing or distribution system.

AGVs can take many different forms, including wheeled or tracked vehicles, carts, pallet jacks, or even drones, depending on the specific application and requirements [17], see Figure 2.1. AGVs are typically controlled by a central computer system that receives information about the location and status of each vehicle and can adjust their routes and schedules to optimise productivity and efficiency [18].

AGVs can offer many benefits to companies that use them, including increased productivity, improved safety, and reduced labour costs. They can also help to optimise material flow and reduce inventory levels, resulting in lower prices and improved customer service.

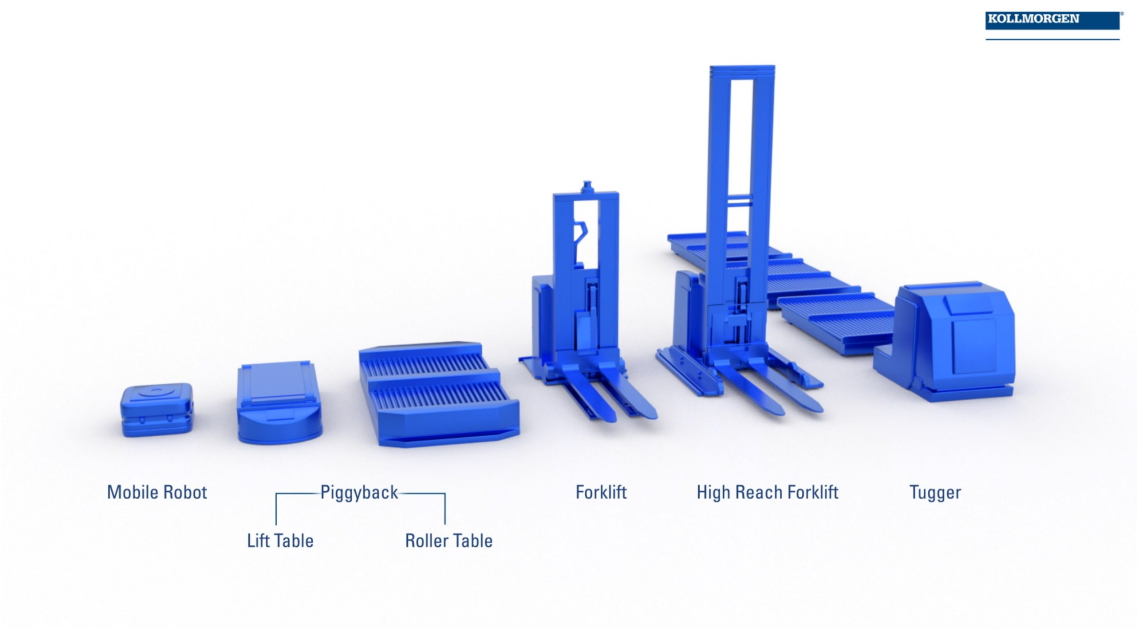


Figure 2.1: Different kinds of AGVs

2.3 General Purpose Solvers

General-purpose solvers are computational tools designed to find solutions to a wide range of problems, often those that can be formulated mathematically or logically. They are typically used in operations research, artificial intelligence, and mathematical optimisation. The general aspect refers to their versatility in handling various problem types instead of being specialised for a specific problem.

A class of mathematical problems is Linear Programming (LP) problems. LP problems are a mathematical model that seeks to maximise or minimise a linear function subject to a set of linear constraints. Here is an example of a simple LP-problem:

$$\begin{aligned} \text{Maximize} \quad & Z = 3x + 2y \\ \text{Subject to} \quad & \\ & x + 2y \leq 14 \\ & 3x - y \leq 6 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$

This LP problem seeks to find the values of x and y that maximize the objective function Z while still satisfying all the constraints. The constraints here express limitations on the variables. In this case, they are both inequality constraints (restricting the values that x and y can take on) and non-negativity constraints (specifying that x and y cannot be negative). This problem can be solved graphically in this simple case but not when the number of variables increases.

Figure 2.2 is a graphical representation of the problem, the blue line is the upper bound of $x + 2y \leq 14$, the orange line is the upper bound of $3x - y \leq 6$. The area between the X-axis, the Y-axis, the blue line and the orange line is the feasible region.

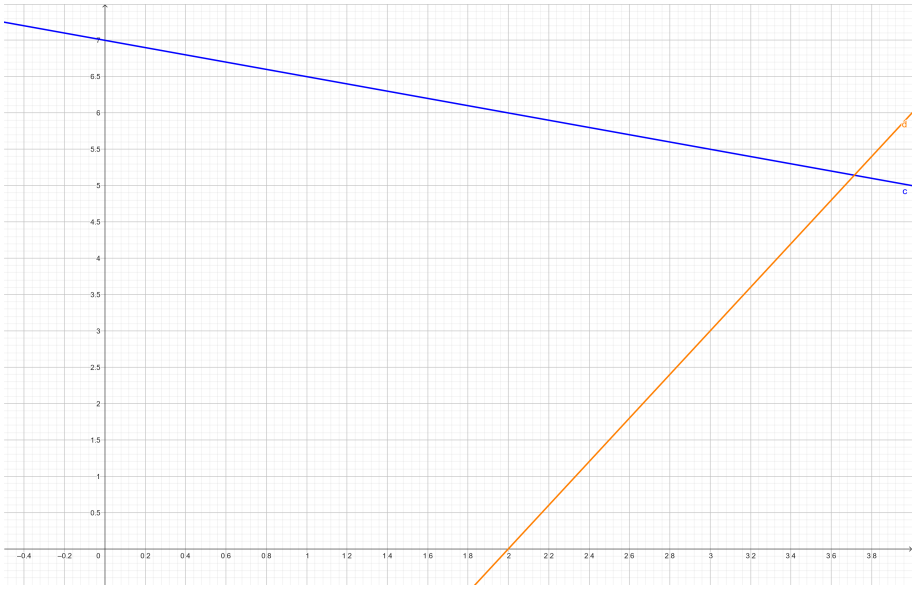


Figure 2.2: Plot of the simple LP example above

A common logic problem is the Satisfiability Problem (SAT) problem. In its simplest form, the Boolean satisfiability problem (also called propositional satisfiability problem and abbreviated SATISFIABILITY or SAT) determines if an interpretation exists that satisfies a given Boolean formula. Here is an example of a simple SAT problem:

Given the Boolean formula (expressed in Conjunctive Normal Form, or CNF):

$$\text{Satisfy:} \quad (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg z \vee x) \tag{2.1}$$

find an assignment of truth values to x, y, and z such that all the clauses of the formula are satisfied. For example, one solution to this problem is $x = \text{True}$, $y = \text{True}$, $z = \text{True}$.

General-purpose solvers include many algorithms to solve these problems,

However, the general nature of these solvers can also be a limitation. While versatile, they may not be as efficient or effective as specific solvers or algorithms designed for a particular class of problems. For instance, a solver built specifically for Finding the shortest paths can be efficiently addressed within polynomial time using the A* algorithm. However, if the same problem is formulated as a Mixed Integer Linear Programming (MILP) problem and addressed by a MILP solver, it could take exponential time in the worst-case scenario. Two well-known general-purpose solvers are Gurobi and Z3.

Gurobi [19] is a commercial optimisation solver that provides high-performance solutions for mathematical programming. It supports a variety of problem types,

including but not limited to linear programming (LP) where the variables are continuous, integer programming (IP) where the variables are integer, and mixed integer linear programming (MILP) where the variables can be continuous or integer mixed in the same problem. The strengths of Gurobi include its robustness, efficiency, and speed, often outperforming other commercial solvers in benchmark tests. It also provides a user-friendly interface and supports various programming languages like Python, C++, and Java. However, its main weakness lies in its commercial nature, which means it is not freely available for all users, and its cost can be prohibitive for some applications.

Z3 [20], on the other hand, is a high-performance Satisfiability Modulo Theories (SMT) solver developed by Microsoft Research. It is used in various fields, including software verification, testing, abstract interpretation, program synthesis, and more. Z3 supports a range of theories, such as propositional logic, first-order logic, and theories of arithmetic, and more. Z3's strengths lie in its wide-ranging logic support, high performance in many applications, and its open-source nature, which allows for wide accessibility and no cost. However, it can be more complex and challenging to use compared to more specialised or user-friendly solvers.

In conclusion, both general-purpose and specific solvers have their strengths and weaknesses. The choice between them depends on the particular requirements of the problem, such as the problem type, required efficiency, available resources, and the level of user expertise.

2.4 Challenges

Navigating the landscape of this study entails confronting a multitude of challenges. Among these, several stand out as particularly significant due to their direct impact on the core objectives of this thesis. These crucial challenges, which form the primary focus of our exploration and analysis, are discussed in the following sections. Each issue presents unique hurdles and implications, demanding distinct strategies and solutions for effective mitigation. Understanding these challenges is crucial for comprehending the intricacies of our problem domain and is pivotal in crafting efficient and effective solutions. Therefore, in the context of this thesis, the following challenges assume heightened importance.

2.4.1 Real-time Computation

Real-time computation presents numerous challenges, including the need for fast processing, efficient algorithms, and low-latency communication [21]. It requires systems to process and analyse data as it is generated, often within strict time constraints. Examples of real-time applications include stock trading [22], autonomous vehicles [23], and air traffic control [24]. Specialised hardware and software solutions are often employed to meet these challenges. However, the unpredictability of real-time tasks and the potential for high-stakes consequences when errors occur make real-time computation a complex and demanding field.

2.4.2 Large-scale Problems

Large-scale problems involve a significant amount of data or numerous variables and constraints, requiring considerable computational resources and advanced algorithms. Examples include simulating global climate models [25], managing large-scale transportation networks [26], and analysing big data sets in genomics [27]. These problems can be computationally expensive, and solving them often necessitates parallel processing, distributed systems, and sophisticated optimisation techniques. Additionally, large-scale problems may suffer from scalability issues, as increasing problem size can lead to exponential growth in required resources.

2.4.3 Complex Constraints

Complex constraints refer to the multiple, often competing requirements and limitations that must be satisfied when solving a problem or designing a system. Examples include optimising energy consumption while maintaining performance in an electric vehicle [28] or scheduling airline flights while considering airport capacity, maintenance, and crew availability [29]. Dealing with complex constraints can be challenging, as it often requires finding a balance between conflicting objectives, managing uncertainty, and navigating an ample search space of potential solutions. Techniques such as multi-objective optimisation, constraint programming, and machine learning can help address these challenges.

Wrap up

This chapter has provided an overview of various critical concepts and challenges that form the foundation of this thesis. At first, it was a discussion introducing the concept of VRPs, highlighting their significance in today's transportation and logistics landscapes and describing some differences between the classical VRP problem and the problem considered in the thesis.

Subsequently, the focus was shifted to AGVs, elaborating on their roles and functional principles within modern warehousing and manufacturing environments.

Much of this chapter was dedicated to understanding General Purpose solvers, computational tools essential for solving complex optimization problems often associated with AGV scheduling. It included insights into these solvers' capabilities and potential limitations, setting the stage for their use in this thesis.

Finally, some specific challenges this thesis aims to address were discussed, such as the complexity of real-time computation, large-scale problem management, and navigating complex constraints. These challenges pose a formidable barrier to deploying AGVs and VRPs effectively. In this regard, this chapter has shed light on these complexities, ensuring a solid grounding for the investigations in the subsequent sections of this thesis.

3

Problem Formulation

This chapter offers a broad-ranging overview of various foundational concepts and components that underpin this study. It begins with a look at the overarching Architecture Overview, providing insights into the structure and interactions of different elements within our system.

Following this, the chapter delves into the System Initialization process. This section sheds light on how the system is set up and readied for operation, describing the preparatory steps and actions needed to launch and operationalise the system.

A thorough examination of Graph theory is then undertaken. As graphs form the basis for understanding and modelling routing problems, this section elucidates their structure, properties, and role in our context.

Then the Order Manager is explored, an essential component responsible for managing and coordinating various tasks within our system. Its functions, responsibilities, and importance in the grand scheme of the system are elaborated.

The concept of Conflicts and Overlaps is explained next. These critical issues can significantly impact the performance and efficiency of the system, and understanding them is vital to practical problem-solving.

A detailed discussion on Paths follows this and how they are computed within the thesis. Path computation is a core process in the study, and this section explores the methods and strategies used to calculate and optimise these paths.

Next, the chapter tackles how Scheduling is used in the thesis. This vital process of coordinating and timing the activities of AGVs is dissected and explained, providing insights into the methods used and the challenges encountered.

Finally, the chapter concludes with a comprehensive description of the Mathematical Models for the Pathing Problem and the Scheduling Problem. These models provide the theoretical foundations for the exploration and problem-solving efforts, and understanding them is crucial for tackling the issues at hand.

3.1 Architecture Overview

The system initiates its process upon receiving two critical pieces of information: the layout of the environment and the locations for the vehicles to be scheduled. Each vehicle's assignment is determined by an order provided by an Order Management System (OMS). The OMS, over which the system has no control, defines

each vehicle's start and goal positions. These start and goal positions can also be randomly assigned for testing purposes.

Once the initial setup is complete, the system begins its primary operations. The system first computes paths for all vehicles. This computation can be based on two strategies: finding shortest paths or paths that minimise overlaps.

Following the path computation, the system schedules the vehicles' movements along the paths that were computed in the last step. This scheduling process aims to ensure a smooth flow of vehicles by avoiding conflicts. This scheduling step is crucial in ensuring efficient vehicle movement. These steps are summarised in Figure 3.1 below. The primary operation is inside the blue box.

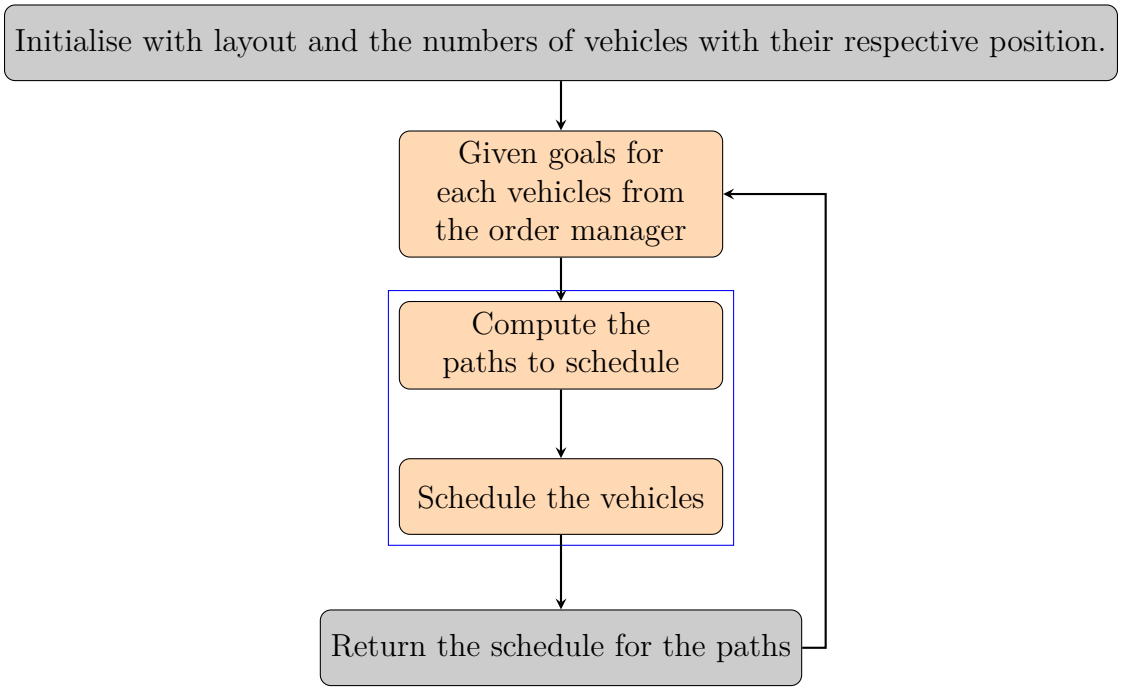


Figure 3.1: Flow of the problem, each box represent a different step.

3.2 System Initialization

The system initialisation begins with two primary inputs:

The first input is a file containing information about the nodes, edges, blockings, and stations. This file serves as a foundation to create a directed and weighted graph, symbolising the physical layout where the vehicles navigate. The nodes represent various locations, the edges depict the possible routes between these locations, blockings indicate places where overlaps can occur, and stations mark particular points of interest.

The second input is a list of (start, goal) positions. Each pair in the list corresponds to a specific vehicle, with the start and goal positions representing the initial location and the vehicle's final destination, respectively. Alternatively, the system can also

provide just the starting positions for the vehicles, with the goal positions being determined by an order manager.

3.3 Graph

A mathematical graph is a fundamental structure used in various branches of mathematics, computer science, and other disciplines to represent relationships between objects [30]. It consists of two main components: nodes and edges. Nodes (or vertices) are the basic units in a graph, representing individual objects or entities. Depending on the graph's context, these can be anything from numbers, people, cities, or even abstract concepts. Edges are the connections or links between these nodes. They represent relationships, interactions, or dependencies between the nodes. Edges can be directed or undirected, and they can have weights assigned to them to quantify the strength of the relationship or the cost associated with the connection.

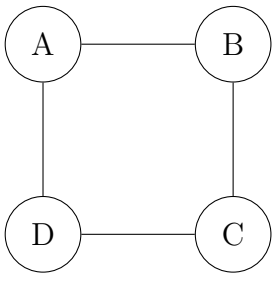
Graphs can be visualised as a collection of points (nodes) connected by lines (edges). In a directed graph, also known as a digraph, edges have a direction, typically represented by arrows. In an undirected graph, edges have no direction and can be traversed in any order.

3.3.1 Example: Undirected and unweighted graph

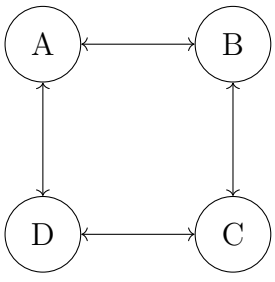
Consider a simple undirected graph with four nodes (A, B, C, and D) and the following edges:

- A - B
- B - C
- C - D
- D - A

This can be drawn in two different ways as seen in Figure 3.2a and Figure 3.2b.



(a) One way to represent an undirected graph



(b) Another way to represent an undirected graph

Figure 3.2: Two ways to draw a undirected and unweighted graph

This graph connects nodes A, B, C, and D by the specified edges, forming a square.

3.3.2 Example: Directed and unweighted graph

Consider a simple directed graph with four nodes (A, B, C, and D) and the following edges:

- A → B
- B → C
- C → D
- D → A

This can be drawn as seen in Figure 3.3.

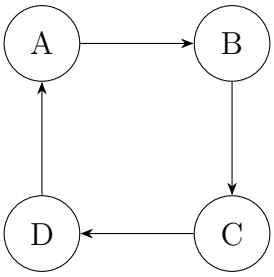


Figure 3.3: Directed and unweighted graph

This graph connects nodes A, B, C, and D by the specified edges, forming a square that can only be traversed in one direction.

3.3.3 Example: Directed and weighted graph

Consider a simple directed graph with four nodes (A, B, C, and D) and the following edges:

- A → B (4)
- B → C (3)
- C → D (2)
- D → A (1)

This can be drawn as seen in Figure 3.4.

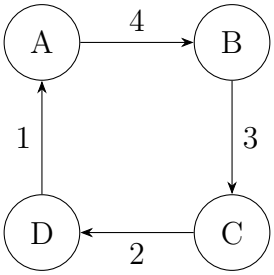


Figure 3.4: Directed and weighted graph

3.3.4 Example: Test layout

Figure 3.5 is the layout used for testing the algorithms and its simplicity helped ensure that the programs worked as intended.

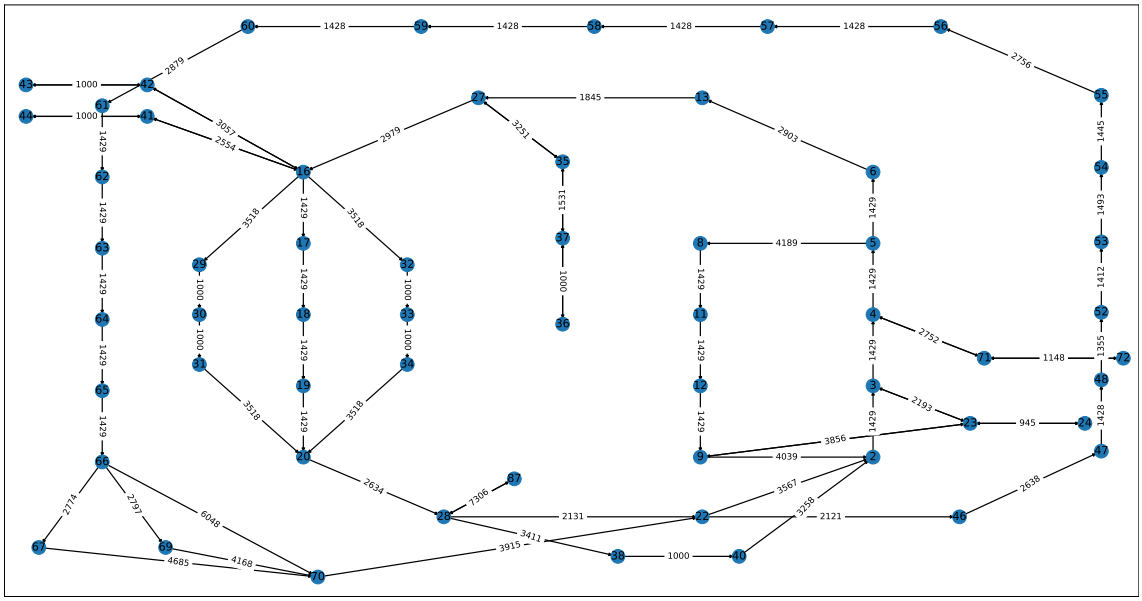


Figure 3.5: Example layout of a weighted, directed graph.

3.4 Order manager

An Order Management System (OMS) is a comprehensive system used in various industries to manage and fulfil sales orders [31]. This digital system tracks orders from inception to delivery and manages the related processes to ensure a smooth operation. The primary functions of an OMS include processing and fulfilling orders, inventory management, and tracking orders' progress.

In the context of AGVs, an OMS assigns tasks or orders to vehicles. For example, an order might involve moving an item from one location to another within a warehouse or manufacturing facility. The OMS considers each vehicle's current status and position and other factors such as task priority, distance to the target, and battery level to determine which vehicle should fulfil which order.

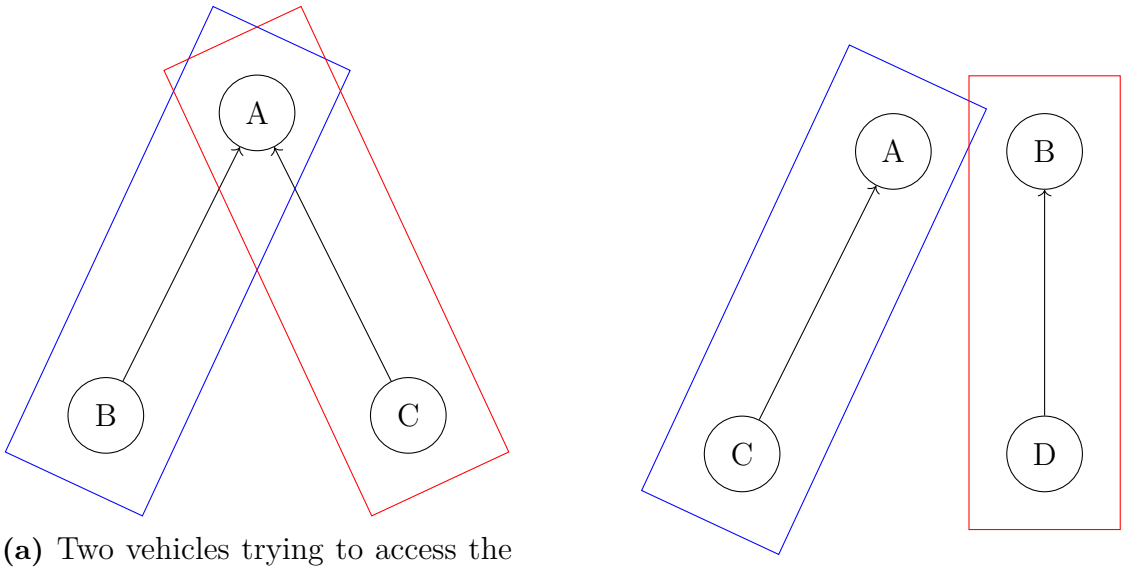
The system continually receives and processes new orders, adjusting vehicle assignments as necessary. It communicates with each vehicle, providing instructions on where and what task to perform. By doing so, the OMS coordinates the movement and activities of multiple vehicles within a facility, ensuring efficient use of resources and minimising conflicts or bottlenecks.

In a real system, the position of each vehicle would be known, and only the goals would be provided, but while testing, all pairs of (start, goal) nodes for each vehicle were randomly chosen such that all vehicles can reach their destinations and that the goal node is different from the starting node. The pairs affect the system's overall performance, and as such multiple different pairs were chosen to avoid bias and

ensure that the results represent the system's behaviour under different conditions. To ensure that the pairs are chosen consistently, a specific seed can be used to retest the same pairs, allowing for a way to redo the same tests again.

3.5 Conflicts and Overlaps

Conflicts can only occur when paths overlap. An overlap occurs when parts of two or more paths overlap. There are three primary ways overlaps can occur that are considered in this report. The first way is that the overlaps can originate from the fact that the nodes are the same in different paths, see Figure 3.6a, or that they are close enough that if two vehicles tried to be on one node each, they would collide, as can be seen in Figure 3.6b.

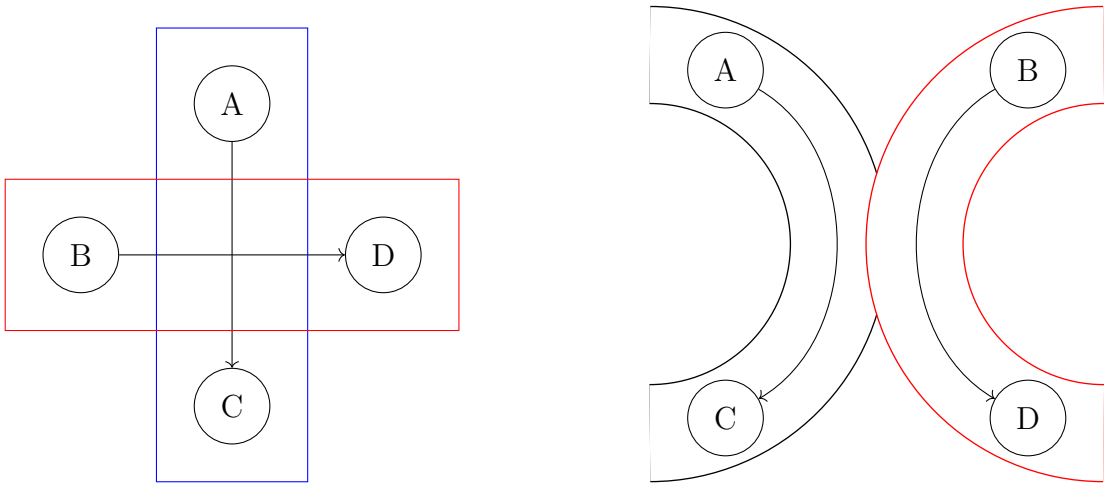


(a) Two vehicles trying to access the same node at the same time

(b) Two nodes being too close

Figure 3.6: Two ways a node and a node overlap can occur

The same can happen for edges, either they cross at some point, see Figure 3.7a, or that they are too close to each other, see Figure 3.7b.



(a) Two edges crossing **(b)** Two edges being too close

Figure 3.7: Two ways an edge and an edge overlap can occur

Another way overlaps can occur is when an edge is too close to a node

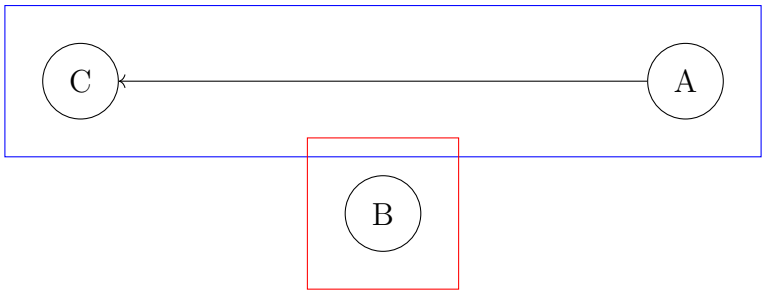


Figure 3.8: One way a node and an edge can overlap

Managing vehicle routing and scheduling conflicts is critical to ensuring efficient and cost-effective operations. By carefully considering the potential sources of conflict and implementing appropriate techniques to mitigate them, vehicle routing systems can achieve better performance and deliver higher customer satisfaction.

Various techniques and algorithms can address these conflicts and overlaps in vehicle routing and scheduling. For example, optimisation algorithms can compute paths that minimise the likelihood of overlaps so that the scheduling can more easily avoid conflicts, this will be discussed in Section 3.6, and the mathematical model for it will be presented in Section 3.8. The conflicts will be dealt with during the scheduling in Section 3.7, and the mathematical model for it will be presented in Section 3.9.

3.6 Pathing

In graph analysis, the shortest path problem is a fundamental concept that involves finding the path with the minimum distance between two nodes in a graph. One common way to solve this problem is to use Dijkstra's algorithm [6], which finds

the shortest path between two nodes in a weighted graph. Another is to use A* [7], an algorithm that, much like Dijkstra's, aims to find the shortest path in a weighted graph. However, A* incorporates a heuristic method to estimate the cost (or distance) from a given node to the goal, thereby guiding its search towards the goal and often solving the problem more efficiently.

Both algorithms, Dijkstra's and A*, operate by exploring the most promising nodes first, based on the accumulated cost from the start node to the current node. This cost is the sum of the weights of the edges traversed. Dijkstra's algorithm systematically explores all paths starting from the source node and extending outwards, always selecting the node with the smallest accumulated cost for expansion.

A*, on the other hand, considers both the accumulated cost and the estimated cost to the goal (the heuristic) when deciding which node to expand next. The heuristic is often the distance from the current node to the goal node, either the euclidean distance or the Manhattan distance depending on how the problem is formulated. This combined cost is denoted as $f(n) = g(n) + h(n)$, where $g(n)$ is the accumulated cost from the start node to the current node, and $h(n)$ is the estimated cost from the current node to the goal. Dijkstra's algorithm can be seen as the special case where the estimated cost $h(n) = 0$. This heuristic-driven approach often allows A* to find the shortest path more quickly while also expanding fewer nodes than Dijkstra's algorithm, especially in larger graphs or when a clear path towards the goal exists.

While Dijkstra's algorithm ensures the shortest path in any graph given non-negative edge weights, A* also guarantees this under the condition that the heuristic function $h(n)$ is admissible – meaning it never overestimates the actual cost to reach the goal – and consistent or monotonic.

NetworkX [32, 33], a Python package for creating, manipulating, and studying complex networks, implements Dijkstra's algorithm to compute shortest paths between pairs of node pairs in a graph. In order to speed up the computation of shortest paths, NetworkX uses Bidirectional Dijkstra's algorithm [34, 35, 36].

Implementing Bidirectional Dijkstra's algorithm will often lead to significant speed improvement, commonly more than double, compared to the standard unidirectional Dijkstra's algorithm [36]. The unidirectional Dijkstra's algorithm operates by expanding nodes outward from the source in a pattern similar to a sphere. This metaphorical sphere's radius eventually matches the shortest path's length. In contrast, the Bidirectional Dijkstra's algorithm simultaneously expands nodes from both the source and the target, resulting in two spheres, each with half the radius of the original sphere. By considering the volumes of these spheres, it becomes evident that the combined volume of the two smaller spheres is half that of the larger sphere, leading to efficiency improvements in the bidirectional approach.

In specific applications, simply finding the shortest path between two nodes in a graph may not be sufficient. For example, if the goal is to schedule vehicles to travel through a layout to serve customers, one must ensure that these vehicles do not simultaneously occupy the same point in space. Therefore, when computing paths for them, one way to reduce the chances of conflicts is to ensure that their paths as little as possible, preferably not at all.

Formulating the path computation problem as an optimisation problem and using Gurobi to solve it, ensures that the computed paths satisfy all constraints and overlaps are minimised, which should result in fewer overlaps than only using the shortest paths. However, compared to NetworkX, using Gurobi requires specialised optimisation knowledge and more computational resources to find a solution.

3.7 Scheduling

In this system, the scheduling output is fundamentally a timeline, indicating when vehicles will occupy and vacate each node, assuming that the transit time between two nodes is equivalent to one time step, this is done by assuming the weights of the edges are one. Each vehicle will advance to the next node in its path as soon as possible if it is not obstructed. The mathematical model used for the scheduling in this thesis will be presented in Section 3.9.

This operational framework can be interpreted in terms of precedence constraints. For a given path, each node's occupation and subsequent vacating essentially set a precedence order, dictating the sequence of movements of the vehicle assigned to such path across the network. A node must be vacated before the subsequent node can be occupied, establishing a chronological order of node access for the vehicles. This mechanism of precedence constraints is integral to the system's efficient functioning, guiding the vehicles' movements and defining constraints to avoid conflicts.

3.8 Mathematical model for the Pathing Problem

In this section, a description of the mathematical model for the Pathing Problem is detailed, with a specific focus on minimizing overlaps. The discussion unfolds by listing the required information, explaining the specific functions used within the model, and outlining their significance. Additionally, the section illustrates the variables used in the model's operation, detailing their roles, potential values, and how they interact within the system. This section explains the mathematical model's structure and functionality in addressing the Pathing Problem.

Sets:

- $\mathcal{N}$: set of nodes
- $\mathcal{E}$: set of edges
- $\mathcal{P}$: set of pairs (start, goal) for each vehicle
- NB_e: set of nodes that block edge e
- EB_e: set of edges that block edge e

Functions:

- P(p): function that returns the pair of nodes associated with index $p \in \mathcal{P}$
- EO(n): function that returns the set of edges originating from node $n \in \mathcal{N}$
- EE(n): function that returns the set of edges ending at node $n \in \mathcal{N}$

Variables:

- M: Is a large number used to indicate if there are more than one conflict

Decision Variables:

- $x_{p,n}$: Boolean variable if a node n is used by a vehicle p
- $y_{p,e}$: Boolean variable if an edge e is used by a vehicle p
- u_n: Boolean variable if a node n is used by a multiple vehicles (not strictly necessary)
- nb_e: Boolean variable if nodes are used that block edge e (not strictly necessary)
- eb_e: Boolean variable if edges are used that block edge e (not strictly necessary)

Model:

$$\min \sum_{n \in \mathcal{N}} u_n + \sum_{p \in \mathcal{P},\ n \in \mathcal{N}} x_{p,n} + \sum_{e \in E} nb_e + \sum_{e \in E} eb_e \tag{3.1}$$

$$\begin{aligned}
\text{s.t.} \quad x_{p,n} &= 1 && \forall p \in \mathcal{P},\ n \in \mathrm{P}(p) && (3.2)\\
\sum_{e \in \mathrm{EO}(n_1)} y_{p,e} &= 1 && \forall p \in \mathcal{P},\ (n_1, n_2) \in \mathrm{P}(p) && (3.3)\\
\sum_{e \in \mathrm{EO}(n_2)} y_{p,e} &= 0 && \forall p \in \mathcal{P},\ (n_1, n_2) \in \mathrm{P}(p) && (3.4)\\
\sum_{e \in \mathrm{EE}(n_2)} y_{p,e} &= 1 && \forall p \in \mathcal{P},\ (n_1, n_2) \in \mathrm{P}(p) && (3.5)\\
\sum_{e \in \mathrm{EE}(n_1)} y_{p,e} &= 0 && \forall p \in \mathcal{P},\ (n_1, n_2) \in \mathrm{P}(p) && (3.6)\\
x_{p,n} &= \sum_{e \in \mathrm{EO}(n)} y_{p,e} && \forall p \in \mathcal{P}, n \in \mathcal{N} \setminus \mathrm{P}(p) && (3.7)\\
x_{p,n} &= \sum_{e \in \mathrm{EE}(n)} y_{p,e} && \forall p \in \mathcal{P}, n \in \mathcal{N} \setminus \mathrm{P}(p) && (3.8)\\
u_n \cdot \mathrm{M} &\geq \sum_{p \in P} x_{p,n} - 1 && \forall n \in \mathcal{N} && (3.9)\\
nb_e \cdot \mathrm{M} &\geq \sum_{p \in P,\ n \in \mathrm{NB}_e} x_{p,n} && \forall e \in \mathcal{E} && (3.10)\\
eb_e \cdot \mathrm{M} &\geq \sum_{p \in P,\ e_2 \in \mathrm{EB}_e} y_{p,e_2} && \forall e \in \mathcal{E} && (3.11)\\
x_{p,n} &\in \{0,1\} && \forall p \in \mathcal{P},\ \forall n \in \mathcal{N} && (3.12)\\
y_{p,e} &\in \{0,1\} && \forall p \in \mathcal{P},\ \forall e \in \mathcal{E} && (3.13)\\
u_n &\in \{0,1\} && \forall n \in \mathcal{N} && (3.14)\\
nb_e &\in \{0,1\} && \forall e \in \mathcal{E} && (3.15)\\
eb_e &\in \{0,1\} && \forall e \in \mathcal{E} && (3.16)
\end{aligned}$$

The objective function (3.1) is to minimise the number of potential overlaps and the length of the paths.

Constraint (3.2) states that all starting nodes and goal nodes are set to 1. Constraint (3.3) states that the number of outgoing edges is set to 1 for the start node, and constraint (3.4) states that they are set to 0 for the goal node. Constraint (3.5) states that the number of incoming edges is set to 0 for the start node, and constraint (3.6) states that they are set to 1 for the goal node. Constraint (3.7) states that for the rest of the nodes, if a node is used, set the number of outgoing edges to 1 and constraint (3.8) states that if a node is used set the number of incoming edges to 1. If an incoming edge is one, the node has to be used, and one outgoing edge has to be used. Constraint (3.9) says if a particular node is used more than once. Constrains (3.10) and (3.11) are not strictly necessary, but they say if some nodes and edges could conflict with edge e. The constraints (3.12), (3.13), (3.14), (3.15), (3.16) set the bounds on the decision variables to be binary.

3.9 Mathematical model for the Scheduling Problem

In this section, a description of the mathematical model for the Scheduling Problem is detailed. The discussion unfolds by listing the required information, explaining the specific functions used within the model, and outlining their significance. Additionally, the section illustrates the variables used in the model's operation, detailing their roles, potential values, and how they interact within the system. This section explains the mathematical model's structure and functionality in addressing the Scheduling Problem.

Sets:

- $\mathcal{P}$: set of paths
- $\mathcal{N}_p$: set of nodes in path p (accessed as n for the node n and i for the index of node at place i)
- $\mathcal{E}_p$: set of edges in path p (accessed as e for the edge e and i for the index of edge at place i)
- NB_e: set of nodes that block edge e
- EB_e: set of edges that block edge e

Decision Variables:

- $v_{p,n}$: represents the time when the vehicle p visits node n
- $l_{p,n}$: represents the time when the vehicle p leaves node n
- $w_{p,e}$: represents the time when the vehicle p visits edge e

Variables:

- $|\mathcal{N}_p|$: The index of the last node in path p
- $|\mathcal{E}_p|$: The index of the last edge in path p

Model:

$$\min \quad \max_{p \in \mathcal{P}} v_{p,|\mathcal{N}_p|} \tag{3.17}$$

$$\text{s.t.} \quad v_{p,n} \geq 0 \qquad \forall p \in \mathcal{P},\ \forall n \in \mathcal{N}_p \tag{3.18}$$

$$l_{p,n} \geq 0 \qquad \forall p \in \mathcal{P},\ \forall n \in \mathcal{N}_p \tag{3.19}$$

$$w_{p,e} \geq 0 \qquad \forall p \in \mathcal{P},\ \forall e \in \mathcal{E}_p \tag{3.20}$$

$$w_{p,i} = l_{p,i} \qquad \forall p \in \mathcal{P},\ \forall i \in \{1, 2, \ldots, |\mathcal{E}_p|\} \tag{3.21}$$

$$v_{p,i+1} = w_{p,i} + 1 \qquad \forall p \in \mathcal{P},\ \forall i \in \{1, 2, \ldots, |\mathcal{E}_p|\} \tag{3.22}$$

$$v_{p,n} \leq l_{p,n} \qquad \forall p \in \mathcal{P},\ \forall n \in \mathcal{N}_p \tag{3.23}$$

$$v_{p_1,n} > l_{p_2,n} \vee v_{p_2,n} > l_{p_1,n} \qquad \forall p_1, p_2 \in \mathcal{P},\ \forall n \in \mathcal{N}_p \tag{3.24}$$

$$w_{p_1,e} > l_{p_2,n} \vee w_{p_1,e} > v_{p_2,n} \qquad \forall p_1, p_2 \in \mathcal{P},\ \forall e \in \mathcal{E}_p,\ \forall n \in NB_e \tag{3.25}$$

$$w_{p_1,e_1} < w_{p_2,e_2} \vee w_{p_1,e_1} > w_{p_2,e_2} \qquad \forall p_1, p_2 \in \mathcal{P},\ \forall e_1 \in \mathcal{E}_p,\ \forall e_2 \in EB_{e_1} \tag{3.26}$$

The objective function (3.17) is to minimise the time when the last vehicle is done with its path.

The constraints (3.18), (3.19), (3.20) set the bounds on the decision variables to being greater than or equal to 0.

Constraint (3.21) states that the outgoing edge is entered immediately when a node is left. Constraint (3.22) states the node is entered one time step after the incoming edge is entered (it takes one time step to traverse an edge). Constraint (3.23) states that leaving a node has to happen after (or at the same time) as entering the node.

Constraint (3.24) states that a node cannot be used by two vehicles simultaneously. One of them has to leave before another enter. Constraint (3.25) states that an edge cannot be used if the nodes that block it are used simultaneously. One has to leave the node or edge before the other enters the node or edge. Constraint (3.26) states that an edge cannot be used if the edges that block it are used simultaneously. One of them has to leave the edge before the other enters the edge.

Wrap up

This chapter overviewed the integral concepts and components pivotal to this research study. It started with examining the Architecture Overview, offering a detailed understanding of the system's structure and the interactions between its various elements.

After this was an analysis of the System Initialization process, explaining the preparatory steps necessary for setting up, launching, and operationalizing the system.

Next, Graph theory was thoroughly explored, given that graphs are foundational for understanding and modelling routing issues. This section illuminated their structure, properties, and role in this context.

Subsequently, the Order Manager was examined, highlighting its crucial role in managing and coordinating tasks within the system. Its responsibilities and its overall significance within the system were discussed.

Following this, the concept of Conflicts and Overlaps was explained, emphasizing their potential to significantly affect the system's performance and efficiency, thereby underscoring the importance of their understanding of practical problem-solving.

After this was a discussion of Paths and the methods employed within the study to compute them. As path computation represents a central process, the techniques and strategies for calculating and optimizing these paths were explored.

Furthermore, the chapter addressed Scheduling within the study, dissecting this critical process of coordinating and timing the activities of AGVs and providing insights into the methodologies used and challenges faced.

To conclude, the chapter offered a detailed description of the Mathematical Models for the Pathing Problem and the Scheduling Problem. These models, which form the theoretical foundation of the research, are essential for understanding and addressing the challenges presented in this study.

4

Evaluation

This chapter presents the various test cases used in this study. Each of these cases represents a unique scenario or set of conditions designed to evaluate the system's performance and the effectiveness of the investigated strategies. Each test case's characteristics, parameters, and relevance are elaborated upon, providing a clear understanding of the diversity of scenarios used for evaluation.

Following this, the chapter shows the result by a series of plots. These graphical representations offer a visual interpretation of the outcomes, allowing an intuitive understanding of the system's behaviour under different conditions.

4.1 Test Definition

Each test commenced with reading in the layout. Following this, the selection process for the start and goal node pairs was attributed to all vehicles. Subsequently, the paths computed that each vehicle would undertake. This path calculation is based on one of two criteria - either focusing on the shortest path or opting for a path that minimises conflicts. Once these paths were computed, the final step involved creating the schedule to have the vehicles avoid conflicts.

In this work, three actual layouts provided by Kollmorgen were utilised. The layouts have been designated small, medium, and large to maintain confidentiality. These layouts are abstracted into the graphs in which the vehicles moved. The number of vehicles selected for each layout was 5, 10, 15, and 20. The number of vehicles is the same as the number of starting and ending points that were randomly yet feasibly selected. The small layout is a graph with 508 nodes and 1038 edges, providing a relatively simple network for the vehicles to navigate. The medium layout, on the other hand, presents a more complex structure, comprising 1591 nodes and 3132 edges. The large layout is the most intricate of the three, with a network of 3308 nodes and 7255 edges, providing the most challenging environment for the pathing and scheduling algorithms. Three distinct methods were employed to determine the best paths for the vehicles. The first method considered the weights of the edges to calculate the shortest paths, while the other used the fewest number of nodes to calculate the shortest paths. All edges were set to a weight of one. The third method diverged from the conventional shortest-path strategy, focusing instead on minimising the number of overlaps between paths. Each layout and path calculation method was tested using five seeds to ensure that any favourable or unfavourable outcomes that might skew the findings were averaged out. The time to find a solution

was measured to evaluate the efficiency of the different methods. It was done under two conditions: optimising and finding the first feasible solution. These two allowed for a comparison between the computation speed and the makespan of the different paths in optimised and non-optimised scheduling scenarios. This was done to see how fast a solution can be found and how good that solution is compared to when the system has time to find a better solution. If the first solution found is good enough but much faster, that could be more beneficial overall. The design of this experiment resulted in a total of 360 unique combinations of different parameters: three different layouts, four numbers of vehicles, three pathing methods, solving for optimality vs solving for feasibility, and five seeds. Each of these combinations was tested, providing a comprehensive dataset for analysis. This high level of variation ensures the robustness of the results. It allows for examining the performance of the different path calculation methods under a wide range of conditions. To further enhance the reliability of the results, each unique combination was run 20 times with the same seed so that statistics could be gathered from the runs. This repetition smooths out any random variations and provides a more accurate estimate of the performance of the different methods. Given these parameters, the study involved a total of 7200 individual runs. Each run was limited to 10 minutes for each part: creating the paths and scheduling the vehicles. There is an exception for the large layout with 20 vehicles where the time limit was set to 30 minutes for the scheduling, but it only ran until it found the first feasible solution. This time constraint ensured the process was not too impractical regarding computational resources and time. The tests were conducted on a computer with an Intel i7-1165G7 CPU operating at 2.80 GHz and 32 GB of RAM.

4.2 Results

The following figures are cactus plots representing the cumulative run time distribution over the set of problem instances. The X-axis represents sorted problem instances, and the Y-axis represents the run time. Each point on the plot stands for a single problem instance. Problem instances are sorted by increasing time and plotted, making the X-coordinate indicate the number of problem instances and the Y-coordinate indicate the total time it took for the first X tests to run. The data in used for the cactus plots are the time when the scheduler is optimising the schedule.

Looking at Figure 4.1, finding shortest unweighted paths, also known as fewest nodes (SP in blue), is the fastest. In contrast, paths computed to minimise overlaps is slightly slower for the most complex layouts but worst for the more simple layouts (PC in purple) and using weighted shortest paths is the slowest (W_SP in red).

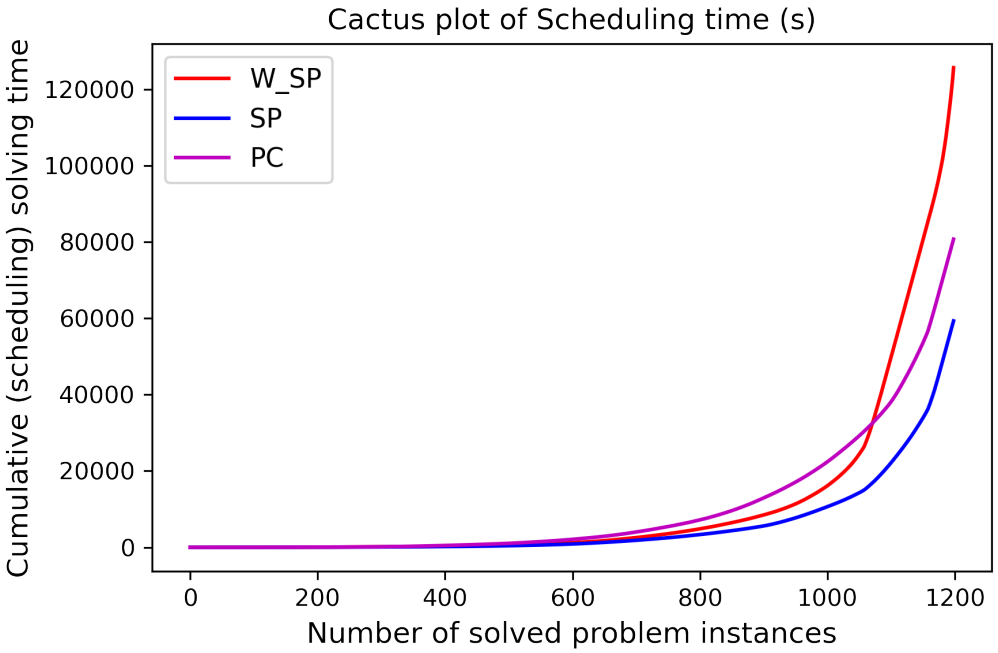


Figure 4.1: Cactus plot of the cumulative scheduling time of running all test, split between the three different variants

Looking at Figure 4.2, the fact that there was a time limit of 10 minutes explains why PC is linear. Using NetworkX to find shortest paths is so fast that, in comparison to Gurobi it is negligible.

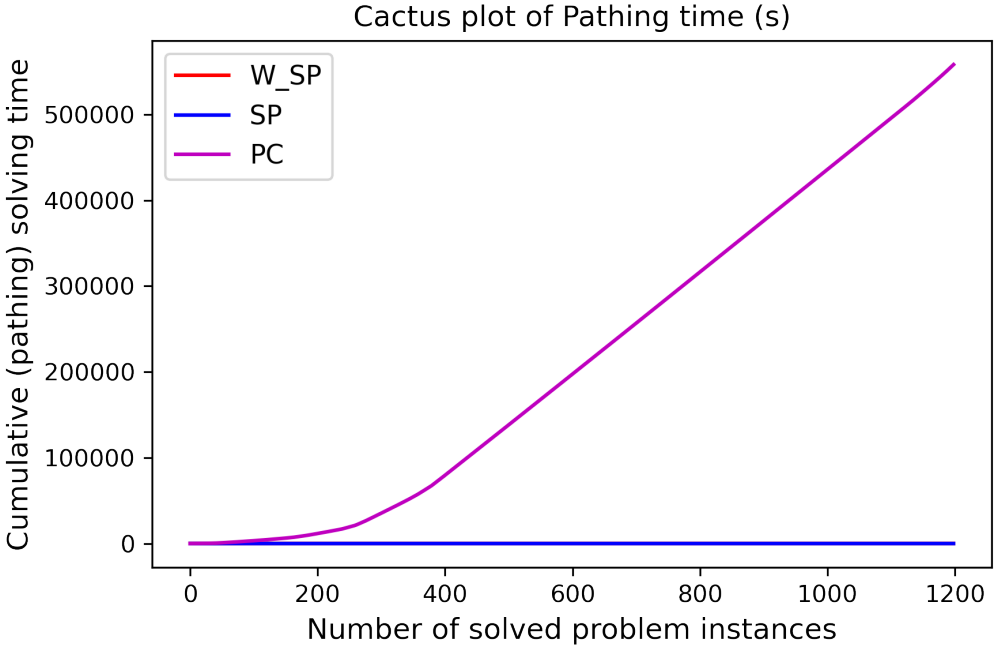


Figure 4.2: Cactus plot of the cumulative pathing time of running all test, split between the three different variants

Looking at the total time, both the time for computing the paths and the schedule in Figure 4.3, it is clear that, while the scheduling using the computation of paths that minimises overlaps might only be slightly worse in the scheduling, it takes so long to compute the paths that in the end it is much faster to use shortest paths.

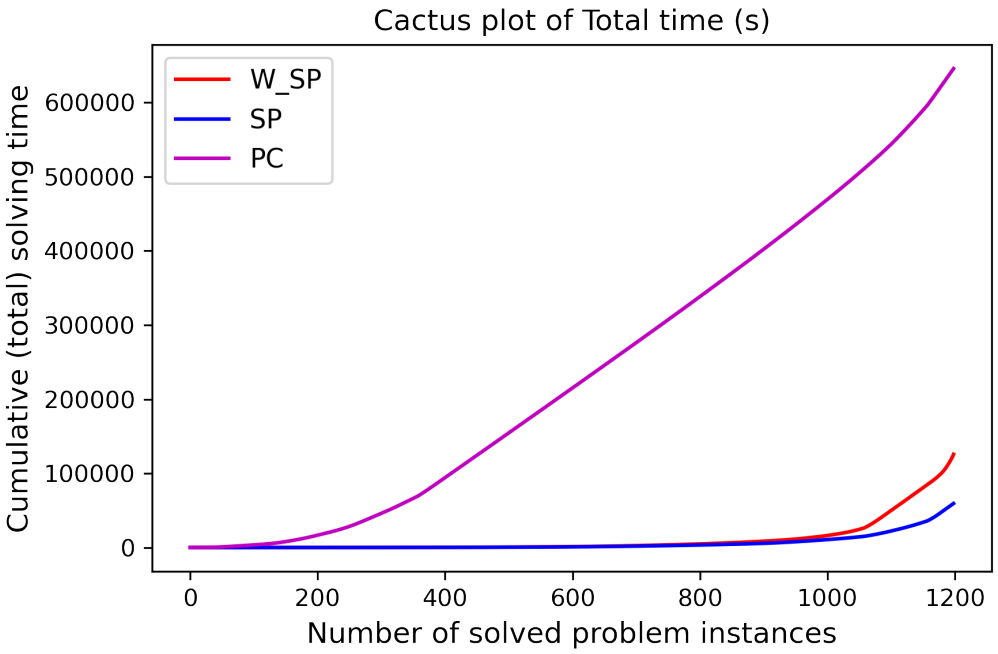


Figure 4.3: Cactus plot of the cumulative (total) time of running all test, split between the three different variants

All data is summarised in Appendix A. There are three tables, one for each layout.

The data shows an inherent correlation between the number of nodes, the objective score, and the scheduling speed. It is observed that a reduced number of nodes yields a lower makespan and a faster scheduling process.

Another result is how good the first solution is compared to the optimal solution. Both in terms how quickly the first solution was found and how much worse the scheduling objective is. Figure 4.4 shows the data as a scatter plot where the x-coordinate is how long the scheduling took and the y-coordinate is the objective for that instance. The colour is used to differentiate between the different way to compute the paths while the shape says if it is the first solution or the optimal solution. When the scheduling time is short, the optimal solution could be as fast or faster in some cases. All figures are in Appendix B.

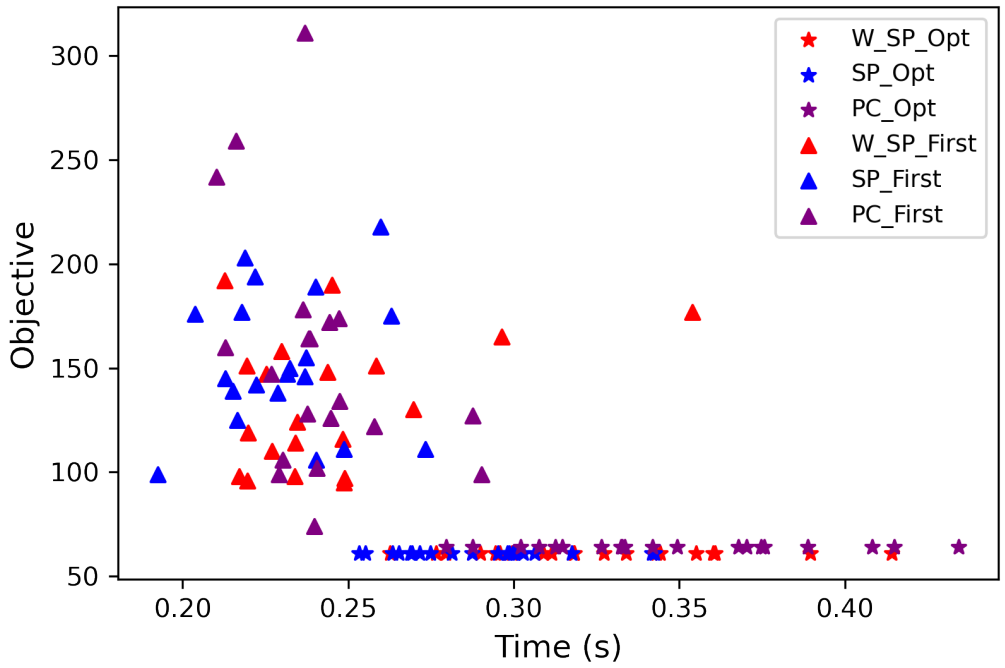


Figure 4.4: Small layout, 5 vehicles, seed 5

Wrap up

Wrapping up this chapter, it provided a presentation of the test cases employed in this research study. Each test case characterizes a distinct scenario or set of conditions to assess the system's performance and the efficacy of the strategies under investigation. Each test case's characteristics, parameters, and relevance were discussed.

After this, the chapter unveils the results via plots and comments. These graphical depictions facilitate a visual comprehension of the data, thereby fostering an intuitive grasp of the system's response and performance under various conditions.

5

Conclusion

This thesis concludes that identifying paths with fewer overlaps presents a possible avenue for improving scheduling efficiency. However, this approach comes with its own set of challenges, including a significantly high computational cost. Furthermore, since this method does not account for conflicts, the resulting performance may be even less satisfactory than initial evaluations suggest.

In the scheduling model used for this study, each edge within the network is presumed equal in length. Consequently, reducing the number of nodes results in a lower scheduling objective. This assumption provides a simplified yet effective way of optimising the scheduling process. However, it is important to remember that this abstraction might overlook some real-world complexities when edge lengths vary. As such, while this approach proves beneficial for this study, exploring models that consider varied edge lengths in future research could be beneficial.

5.1 Further Research and Development

There are multiple ways to continue this study. The first is that the outcomes might differ significantly if the actual weights of the edges were considered instead of assuming all weights are one (using timesteps) in the scheduling. This approach could reflect more accurately the real-world constraints and complexities encountered during the scheduling process. Consequently, it could enhance the overall efficiency and optimisation of the scheduling system.

The second is that multiple paths could have the same length or number of overlaps. If these paths were chosen, the outcome might have varied. As Dijkstra's algorithm returns the first solution, it is not easy to find multiple paths that are as good, and as Gurobi is a black box, it is challenging to investigate. However, since there could be paths that would be better, it could be attractive to investigate further.

The third is that the method employed for computing overlaps in Gurobi does not result in fewer conflicts, as the current method does not track when the overlaps occur. For the moment, the calculation of overlaps checks if the same node or edge is used in more than one path. Having ten potential overlaps, which do not cause conflicts, is preferable to having a single overlap that would cause an actual conflict. Currently, Gurobi would return the paths with one conflicts as it does not know about this and only minimises the number of overlaps. If an improved method of calculating overlaps that would reduce the conflicts were devised and implemented,

it could yield superior results. This enhancement could further reduce the scheduling time and makespan of the scheduling.

The fourth is that this study evaluated the first solution, and the solution was derived after ten minutes. However, examining the rate at which the solution improves over time might be attractive. As previously said, an SMT solver can find a good enough solution regarding the cost function in a relatively short period. This analysis could provide valuable insights into the performance and progression of the optimisation process, offering a more comprehensive understanding of the system's dynamics.

5.2 Ethics and Sustainability

One ethical concern is the potential impact on employment. While AGVs can improve efficiency and reduce labour costs, they can also lead to job displacement or the need for workers to be retrained in new roles. It is essential to consider the potential impacts on workers and take steps to minimise any adverse effects, for instance, by setting up job retraining programs or other forms of support.

Another ethical issue is the potential for AGVs to pose a safety risk to workers and other individuals in the facility. Proper safety protocols and training should be in place to minimise the risk of accidents or injuries.

Regarding sustainability, AGVs can help reduce energy consumption and emissions by streamlining logistics and minimising human transportation needs. However, the production and disposal of AGVs can also have an environmental impact, so it is essential to consider the entire life cycle of the vehicles and take steps to minimise their environmental footprint. This could include using energy-efficient AGVs, designing for recyclability, and implementing proper disposal procedures.

Bibliography

[1] Iris FA Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, 2006.

[2] David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156, 1991.

[3] Michael L Pinedo. *Scheduling*, volume 29. Springer, 2012.

[4] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In *Lecture Notes in Computer Science*, pages 23–36. Springer Berlin Heidelberg, 2009.

[5] Sabino Francesco Roselli, Kristofer Bengtsson, and Knut Akesson. SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, aug 2018.

[6] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[7] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[8] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Cohen, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019.

[9] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

[10] Sabino Francesco Roselli, Per-Lage Götvall, and Fabian. A compositional algorithm for the conflict-free electric vehicle routing problem. *IEEE Transactions on Automation Science and Engineering*, 19(3):1405–1421, 2022.

[11] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[12] Ted K Ralphs, Leonid Kopman, William R Pulleyblank, and Leslie E Trotter. On the capacitated vehicle routing problem. *Mathematical programming*, 94:343–359, 2003.

[13] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[14] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290, 1994.

[15] Barrie M Baker and MA1951066 Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[16] John E Bell and Patrick R McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced engineering informatics*, 18(1):41–48, 2004.

[17] Tuan Le-Anh and MBM De Koster. A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23, 2006.

[18] Matthias De Ryck, Mark Versteyhe, and Frederik Debrouwere. Automated guided vehicle systems, state-of-the-art control algorithms and techniques. *Journal of Manufacturing Systems*, 54:152–173, 2020.

[19] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[20] Leonardo De Moura and Nikolaj Bjørner. Z3 theorem prover. `https://github.com/Z3Prover/z3`, 2023.

[21] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.

[22] Jigar Patel, Sahil Shah, Priyank Thakkar, and K Kotecha. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications*, 42(1):259–268, 2015.

[23] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[24] Will Meilander, Mingxian Jin, and Johnnie Baker. Tractable real-time air traffic control automation. pages 477–483, 01 2002.

[25] Jianduo Li, Chiyuan Miao, Wei Wei, Guo Zhang, Lijuan Hua, Yueli Chen, and Xiaoxiao Wang. Evaluation of cmip6 global climate models for simulating land surface energy and water fluxes during 1979–2014. *Journal of Advances in Modeling Earth Systems*, 13(6):e2021MS002515, 2021. e2021MS002515 2021MS002515.

[26] Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, 17(4):818, 2017.

[27] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha,

and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015.

[28] Samuel Pelletier, Ola Jabali, and Gilbert Laporte. The electric vehicle routing problem with energy consumption uncertainty. *Transportation Research Part B: Methodological*, 126:225–255, 2019.

[29] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, 2017.

[30] Douglas Brent West. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[31] IBM. What is order management? *IBM*, 2021.

[32] NetworkX. Networkx - network analysis in python, 2023.

[33] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gäel Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, 2008.

[34] Gintaras Vaira and Olga Kurasova. Parallel bidirectional dijkstra's shortest path algorithm. *Databases and Information Systems VI, Frontiers in Artificial Intelligence and Applications*, 224:422–435, 2011.

[35] NetworkX. Networkx - shortest_path, 2023.

[36] NetworkX. Networkx - bidirectional_dijkstra, 2023.

A

Appendix 1

Each cell has (min, mean, max) over five different (start/goal) nodes for each vehicle, where each test was run 20 times.

The ID column keeps track of which variables are used in the tests.

- First digit is the size of the layout (1: Small, 2: Medium, 3: Large)
- Second digit is the number of vehicles (1: 5, 2: 10, 3: 15, 4: 20)
- Third digit is if Z3 is optimising or taking the first solution (1: Optimise, 2: First)
- Fourth digit is which method was used for computing the paths (1: Weighted Dijkstra, 2: Unweighted Dijkstra, 3: Gurobi)

The Path time (s) is the time it took to compute the paths using Dijkstra or Gurobi, specified in seconds.

The Overlaps is the objective function for Gurobi, it is the number of overlaps as calculated by Gurobi. The same calculation was done for the Dijkstra paths to see what they would have had for the score. This calculation is done by checking, for each value in all paths, does it have any overlap in the other following paths.

The #Nodes is the total number of nodes in the paths.

The #Conflicts is the total number of conflicts. This is done by looking at the first value in all paths and checking if they conflict, then looking at the second value and so on.

The Z3 time (s) is the time it took to set up and compute the schedule using Z3, specified in seconds.

The Objective Z3 is the makespan from the Z3 calculation, which means the step when the latest vehicle is at its destination.

The Total time (s) is the Path time (s) + Z3 time (s) + the time to do all other calculations.

ID	Path time (s)	Overlaps	#Nodes	#Conflicts	Z3 time (s)	Objective Z3	Total time (s)
1111	(0.001, 0.001, 0.004)	(624, 770.4, 974)	(148, 194.6, 262)	(129, 174.4, 256)	(0.1, 0.2, 0.4)	(44, 56.2, 66)	(0.1, 0.2, 0.4)
1112	(0.0, 0.001, 0.002)	(626, 799.4, 1009)	(147, 190.2, 260)	(115, 162.2, 216)	(0.1, 0.2, 0.3)	(44, 53.4, 61)	(0.1, 0.2, 0.4)
1113	(7.4, 145.1, 600.3)	(620, 742.0, 971)	(174, 202.8, 269)	(129, 182.8, 263)	(0.1, 0.3, 0.6)	(57, 63.6, 69)	(7.9, 145.7, 601.1)
1121	(0.001, 0.001, 0.004)	(624, 770.4, 974)	(148, 194.6, 262)	(129, 174.4, 256)	(0.1, 0.2, 0.4)	(45, 109.0, 192)	(0.1, 0.2, 0.4)
1122	(0.0, 0.001, 0.004)	(626, 799.4, 1009)	(147, 190.2, 260)	(115, 162.2, 216)	(0.1, 0.2, 0.3)	(46, 109.5, 218)	(0.1, 0.2, 0.3)
1123	(8.5, 148.6, 600.6)	(620, 742.0, 971)	(174, 202.8, 269)	(129, 182.8, 263)	(0.2, 0.2, 0.4)	(69, 112.0, 311)	(8.9, 149.2, 601.3)
1211	(0.001, 0.002, 0.007)	(1102, 1288.8, 1544)	(293, 394.0, 502)	(279, 383.6, 529)	(0.3, 2.5, 9.7)	(57, 65.4, 75)	(0.4, 2.5, 9.7)
1212	(0.001, 0.001, 0.003)	(1089, 1259.4, 1567)	(286, 382.4, 480)	(264, 363.4, 481)	(0.3, 2.0, 6.4)	(56, 64.0, 75)	(0.3, 2.1, 6.4)
1213	(38.8, 191.9, 376.1)	(984, 1170.0, 1353)	(296, 400.4, 529)	(277, 390.2, 561)	(0.4, 5.3, 19.4)	(57, 69.0, 91)	(42.9, 197.7, 377.3)
1221	(0.001, 0.003, 0.007)	(1102, 1288.8, 1544)	(293, 394.0, 502)	(279, 383.6, 529)	(0.3, 0.5, 0.8)	(117, 197.4, 466)	(0.3, 0.6, 0.8)
1222	(0.001, 0.001, 0.004)	(1089, 1259.4, 1567)	(286, 382.4, 480)	(264, 363.4, 481)	(0.2, 0.5, 0.8)	(110, 201.6, 549)	(0.3, 0.5, 0.8)
1223	(47.7, 198.6, 394.6)	(984, 1170.0, 1353)	(296, 400.4, 529)	(277, 390.2, 561)	(0.3, 0.6, 1.1)	(111, 215.2, 942)	(48.9, 199.8, 395.7)
1311	(0.002, 0.003, 0.006)	(1542, 1701.8, 1985)	(497, 586.8, 750)	(497, 597.8, 762)	(1.7, 14.1, 47.8)	(63, 76.0, 99)	(1.7, 14.1, 47.8)
1312	(0.001, 0.002, 0.004)	(1531, 1681.6, 1936)	(482, 560.8, 716)	(480, 582.8, 744)	(1.6, 11.0, 42.9)	(63, 69.8, 75)	(1.6, 11.0, 42.9)
1313	(55.1, 373.3, 599.7)	(1396, 1516.4, 1712)	(524, 661.4, 856)	(529, 707.6, 926)	(7.1, 40.6, 133.6)	(86, 89.4, 92)	(73.1, 414.7, 625.8)
1321	(0.002, 0.006, 0.016)	(1542, 1701.8, 1985)	(497, 586.8, 750)	(497, 597.8, 762)	(0.6, 1.2, 1.9)	(134, 273.6, 573)	(0.7, 1.2, 1.9)
1322	(0.001, 0.003, 0.009)	(1531, 1681.6, 1936)	(482, 560.8, 716)	(480, 582.8, 744)	(0.5, 1.1, 2.1)	(133, 265.6, 714)	(0.6, 1.2, 2.1)
1323	(53.1, 384.9, 599.6)	(1396, 1516.4, 1712)	(524, 661.4, 856)	(529, 707.6, 926)	(0.8, 1.9, 3.7)	(152, 250.3, 362)	(55.2, 387.9, 602.4)
1411	(0.003, 0.004, 0.008)	(1717, 1948.2, 2112)	(603, 763.4, 859)	(659, 828.8, 928)	(13.8, 53.5, 114.8)	(69, 80.4, 94)	(13.9, 53.5, 114.8)
1412	(0.001, 0.003, 0.006)	(1719, 1956.2, 2066)	(601, 736.6, 826)	(655, 796.2, 904)	(12.1, 40.7, 85.9)	(69, 71.6, 73)	(12.1, 40.8, 85.9)
1413	(44.2, 154.0, 385.1)	(1541, 1727.8, 1858)	(660, 821.6, 949)	(789, 948.4, 1083)	(22.5, 96.3, 201.0)	(71, 78.8, 84)	(118.9, 251.4, 419.1)
1421	(0.003, 0.004, 0.007)	(1717, 1948.2, 2112)	(603, 763.4, 859)	(659, 828.8, 928)	(0.9, 1.4, 2.1)	(152, 291.9, 497)	(0.9, 1.4, 2.1)
1422	(0.001, 0.003, 0.008)	(1719, 1956.2, 2066)	(601, 736.6, 826)	(655, 796.2, 904)	(0.8, 1.3, 2.0)	(160, 271.9, 455)	(0.9, 1.3, 2.0)
1423	(60.8, 197.0, 599.6)	(1546, 1730.6, 1858)	(660, 821.8, 949)	(789, 948.7, 1083)	(1.5, 2.8, 5.5)	(155, 267.2, 383)	(63.5, 201.3, 604.4)

Table A.1: Result from the small layout.

ID	Path time (s)	Overlaps	#Nodes	#Conflicts	Z3 time (s)	Objective Z3	Total time (s)
2111	(0.002, 0.007, 0.019)	(1714, 2276.2, 2634)	(150, 263.2, 324)	(133, 221.2, 259)	(0.1, 0.4, 0.8)	(50, 70.4, 84)	(0.1, 0.5, 0.9)
2112	(0.001, 0.004, 0.009)	(1662, 2182.6, 2435)	(138, 220.6, 263)	(126, 197.6, 254)	(0.1, 0.4, 1.1)	(50, 55.6, 61)	(0.1, 0.5, 1.2)
2113	(598.4, 635.4, 711.0)	(1552, 1949.6, 2093)	(153, 264.2, 324)	(137, 228.6, 272)	(0.1, 0.7, 2.0)	(55, 69.7, 82)	(601.1, 639.1, 716.5)
2121	(0.002, 0.011, 0.055)	(1714, 2276.2, 2634)	(150, 263.2, 324)	(133, 221.2, 259)	(0.1, 0.4, 0.7)	(63, 132.4, 554)	(0.1, 0.5, 0.8)
2122	(0.001, 0.005, 0.013)	(1662, 2182.6, 2435)	(138, 220.6, 263)	(126, 197.6, 254)	(0.1, 0.4, 0.7)	(50, 98.4, 181)	(0.1, 0.5, 0.8)
2123	(598.3, 632.0, 684.8)	(1552, 1951.2, 2093)	(153, 265.0, 324)	(137, 229.0, 272)	(0.1, 0.6, 1.0)	(55, 148.2, 413)	(600.9, 636.3, 689.6)
2211	(0.006, 0.008, 0.015)	(3068, 3395.0, 3659)	(466, 500.6, 574)	(467, 478.8, 505)	(0.6, 2.4, 12.4)	(69, 77.2, 85)	(0.7, 2.5, 12.4)
2212	(0.002, 0.003, 0.006)	(3023, 3276.8, 3557)	(398, 409.4, 421)	(389, 403.0, 418)	(0.6, 2.0, 4.4)	(52, 58.4, 64)	(0.6, 2.1, 4.5)
2213	(597.5, 598.7, 602.6)	(2553, 2779.8, 3079)	(443, 473.6, 540)	(423, 479.5, 563)	(3.1, 5.9, 11.9)	(58, 75.7, 100)	(604.7, 607.6, 613.3)
2221	(0.006, 0.008, 0.031)	(3068, 3395.0, 3659)	(466, 500.6, 574)	(467, 478.8, 505)	(0.4, 0.6, 1.1)	(178, 301.5, 561)	(0.5, 0.7, 1.1)
2222	(0.002, 0.004, 0.008)	(3023, 3276.8, 3557)	(398, 409.4, 421)	(389, 403.0, 418)	(0.4, 0.5, 0.7)	(82, 163.8, 304)	(0.4, 0.6, 0.8)
2223	(596.8, 598.3, 599.4)	(2600, 2804.6, 3079)	(443, 477.2, 540)	(423, 484.2, 563)	(0.5, 0.7, 1.0)	(123, 205.8, 349)	(601.8, 602.2, 603.2)
2311	(0.006, 0.01, 0.017)	(3588, 3877.6, 4062)	(521, 647.0, 720)	(528, 666.0, 747)	(1.0, 11.7, 60.1)	(63, 78.0, 88)	(1.0, 11.8, 60.2)
2312	(0.003, 0.004, 0.011)	(3691, 3915.2, 4208)	(461, 541.8, 596)	(525, 587.6, 677)	(0.8, 9.5, 22.3)	(51, 61.6, 70)	(0.8, 9.5, 22.3)
2313	(597.1, 597.6, 598.0)	(3238, 3413.8, 3714)	(516, 641.0, 735)	(530, 672.4, 745)	(3.1, 22.7, 57.1)	(63, 79.0, 96)	(605.0, 624.4, 658.8)
2321	(0.006, 0.012, 0.04)	(3588, 3877.6, 4062)	(521, 647.0, 720)	(528, 666.0, 747)	(0.7, 1.1, 1.9)	(150, 314.4, 842)	(0.7, 1.2, 2.0)
2322	(0.003, 0.005, 0.011)	(3691, 3915.2, 4208)	(461, 541.8, 596)	(525, 587.6, 677)	(0.5, 0.8, 1.2)	(107, 187.8, 301)	(0.6, 0.8, 1.2)
2323	(596.8, 597.7, 598.0)	(3238, 3406.6, 3714)	(516, 644.2, 735)	(530, 673.4, 745)	(0.6, 1.0, 1.7)	(128, 225.0, 374)	(602.2, 602.6, 603.4)
2411	(0.009, 0.016, 0.04)	(4319, 4582.4, 4968)	(746, 874.8, 1025)	(799, 907.8, 1074)	(9.9, 25.1, 58.2)	(66, 78.6, 92)	(10.0, 25.2, 58.3)
2412	(0.004, 0.007, 0.017)	(4002, 4397.8, 4777)	(654, 744.2, 848)	(742, 819.8, 949)	(12.9, 40.3, 115.0)	(59, 65.4, 71)	(12.9, 40.3, 115.1)
2413	(595.9, 596.7, 597.2)	(3601, 3824.2, 4214)	(723, 875.8, 1070)	(802, 977.6, 1255)	(20.3, 83.0, 191.0)	(65, 84.4, 101)	(622.4, 685.2, 793.2)
2421	(0.009, 0.017, 0.041)	(4319, 4582.4, 4968)	(746, 874.8, 1025)	(799, 907.8, 1074)	(0.9, 3.0, 9.9)	(153, 360.4, 738)	(1.0, 3.0, 10.0)
2422	(0.004, 0.007, 0.02)	(4002, 4397.8, 4777)	(654, 744.2, 848)	(742, 819.8, 949)	(1.0, 1.5, 2.3)	(132, 222.2, 423)	(1.1, 1.5, 2.4)
2423	(594.4, 596.2, 597.4)	(3601, 3824.2, 4214)	(723, 875.8, 1070)	(802, 977.6, 1255)	(1.3, 8.4, 51.8)	(155, 349.4, 1649)	(603.7, 611.2, 654.5)

Table A.2: Result from the medium layout.

ID	Path time (s)	Overlaps	#Nodes	#Conflicts	Z3 time (s)	Objective Z3	Total time (s)
3111	(0.007, 0.014, 0.047)	(3259, 4213.2, 5094)	(312, 425.8, 528)	(256, 347.6, 489)	(1.2, 3.6, 7.7)	(121, 158.0, 201)	(1.4, 3.8, 7.9)
3112	(0.001, 0.002, 0.003)	(3356, 4102.6, 4856)	(177, 245.2, 298)	(142, 214.2, 258)	(0.3, 0.6, 1.4)	(66, 86.4, 101)	(0.4, 0.7, 1.5)
3113	(199.9, 517.0, 596.9)	(3153, 3852.0, 4437)	(206, 273.6, 314)	(169, 235.2, 273)	(0.4, 0.9, 2.5)	(80, 96.2, 109)	(206.6, 524.6, 605.4)
3121	(0.007, 0.013, 0.044)	(3259, 4213.2, 5094)	(312, 425.8, 528)	(256, 347.6, 489)	(0.5, 0.8, 1.1)	(151, 272.9, 344)	(0.6, 0.9, 1.3)
3122	(0.002, 0.002, 0.004)	(3356, 4102.6, 4856)	(177, 245.2, 298)	(142, 214.2, 258)	(0.2, 0.3, 0.4)	(79, 153.7, 236)	(0.3, 0.4, 0.5)
3123	(191.1, 515.7, 597.4)	(3153, 3851.8, 4437)	(206, 273.4, 305)	(169, 235.2, 273)	(0.2, 0.4, 0.6)	(96, 186.0, 469)	(196.9, 522.4, 604.2)
3211	(0.012, 0.02, 0.067)	(4781, 5367.4, 5884)	(672, 801.8, 985)	(648, 839.0, 1126)	(13.3, 73.4, 232.3)	(166, 191.6, 223)	(13.5, 73.5, 232.4)
3212	(0.002, 0.005, 0.012)	(4771, 5334.0, 5780)	(403, 514.6, 682)	(380, 530.8, 735)	(2.0, 13.9, 48.0)	(76, 103.0, 154)	(2.0, 14.0, 48.2)
3213	(591.6, 593.0, 593.9)	(4475, 5007.2, 5399)	(414, 558.4, 753)	(402, 584.2, 811)	(4.1, 20.5, 74.3)	(88, 116.2, 165)	(607.5, 624.0, 677.9)
3221	(0.011, 0.019, 0.057)	(4781, 5367.4, 5884)	(672, 801.8, 985)	(648, 839.0, 1126)	(1.1, 2.1, 4.3)	(281, 416.4, 585)	(1.2, 2.2, 4.7)
3222	(0.003, 0.005, 0.012)	(4771, 5334.0, 5780)	(403, 514.6, 682)	(380, 530.8, 735)	(0.7, 1.3, 2.2)	(122, 213.5, 347)	(0.8, 1.4, 2.3)
3223	(589.1, 592.8, 593.8)	(4475, 5007.2, 5399)	(414, 558.4, 753)	(402, 584.2, 811)	(0.7, 1.2, 2.1)	(143, 247.1, 473)	(604.1, 604.7, 606.0)
3311	(0.012, 0.027, 0.066)	(6076, 6542.4, 7089)	(986, 1226.4, 1557)	(1023, 1324.0, 1732)	(24.2, 320.8, 611.1)	(183, 269.5, 651)	(24.5, 321.0, 611.3)
3312	(0.003, 0.006, 0.016)	(5879, 6315.0, 6667)	(538, 761.8, 893)	(607, 842.4, 1072)	(13.5, 109.2, 267.2)	(94, 112.8, 128)	(13.6, 109.3, 267.3)
3313	(589.7, 591.0, 593.3)	(5564, 5888.8, 6282)	(585, 802.0, 918)	(650, 885.4, 1140)	(15.6, 135.0, 408.1)	(97, 115.0, 129)	(620.7, 740.0, 1013.0)
3321	(0.012, 0.027, 0.071)	(6076, 6542.4, 7089)	(986, 1226.4, 1557)	(1023, 1324.0, 1732)	(11.1, 184.6, 605.4)	(368, 950.6, 2418)	(11.3, 184.8, 605.7)
3322	(0.003, 0.006, 0.016)	(5879, 6315.0, 6667)	(538, 761.8, 893)	(607, 842.4, 1072)	(0.9, 2.3, 3.8)	(210, 330.7, 510)	(1.1, 2.5, 3.9)
3323	(589.6, 591.0, 593.2)	(5564, 5888.8, 6282)	(585, 802.0, 918)	(650, 885.4, 1140)	(1.0, 2.4, 3.9)	(194, 334.0, 544)	(606.1, 607.4, 609.1)
3411	(0.025, 0.033, 0.063)	(6749, 7614.6, 8182)	(1465, 1729.4, 2086)	(1645, 1870.2, 2291)	(405.3, 748.4, 2148.7)	(176, 686.7, 3295)	(405.5, 748.8, 2149.1)
3412	(0.007, 0.009, 0.013)	(6380, 7202.0, 7747)	(909, 1110.0, 1333)	(1018, 1247.2, 1597)	(132.1, 363.1, 613.0)	(96, 139.1, 284)	(132.3, 363.4, 613.4)
3413	(578.3, 587.5, 588.4)	(6086, 6858.5, 7304)	(956, 1152.4, 1375)	(1040, 1281.2, 1611)	(99.8, 396.0, 611.7)	(101, 148.6, 293)	(706.7, 1003.0, 1219.3)
3421	(0.027, 0.036, 0.06)	(6749, 7614.6, 8182)	(1465, 1729.4, 2086)	(1645, 1870.2, 2291)	(129.9, 614.4, 1802.8)	(507, 1159.3, 3956)	(130.0, 623.9, 1803.1)
3422	(0.007, 0.009, 0.018)	(6380, 7202.0, 7747)	(909, 1110.0, 1333)	(1018, 1247.2, 1597)	(4.8, 74.6, 249.6)	(291, 569.1, 1268)	(5.0, 74.9, 250.0)
3423	(574.1, 587.1, 588.6)	(6086, 6858.4, 7304)	(956, 1152.2, 1375)	(1040, 1280.9, 1611)	(4.3, 96.3, 380.9)	(312, 623.0, 1886)	(611.0, 703.5, 991.5)

Table A.3: Result from the large layout.

B

Appendix 2

Each figure in this chapter is a scatter plot where the x-coordinate is how long the scheduling took and the y-coordinate is the objective for that instance. The colour is used to differentiate between the different ways to compute the paths while the shape says if it is the first solution or the optimal solution.

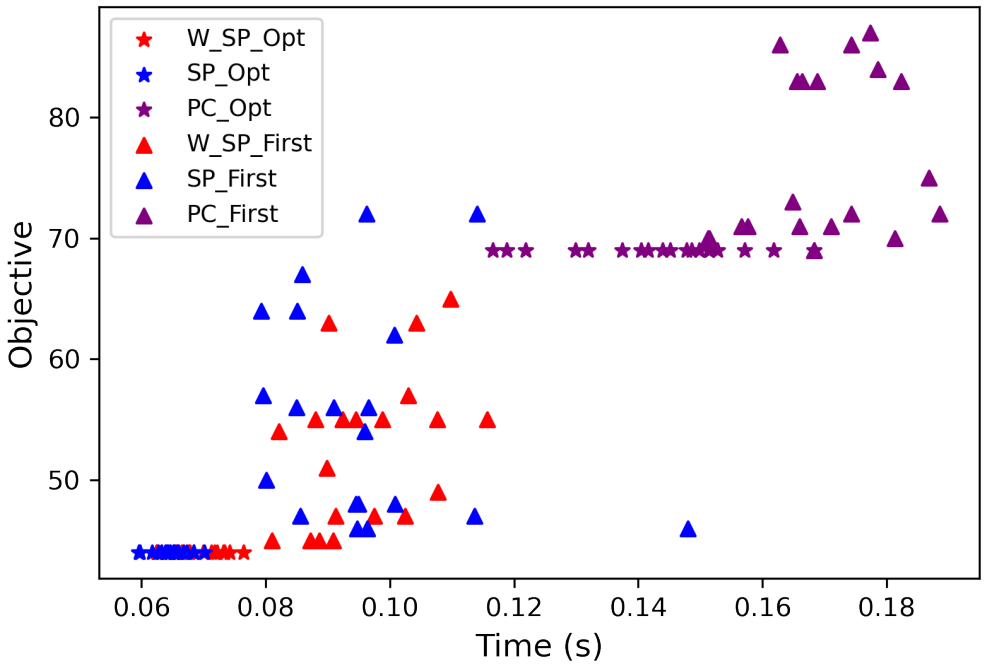


Figure B.1: Small layout, 5 vehicles, seed 1

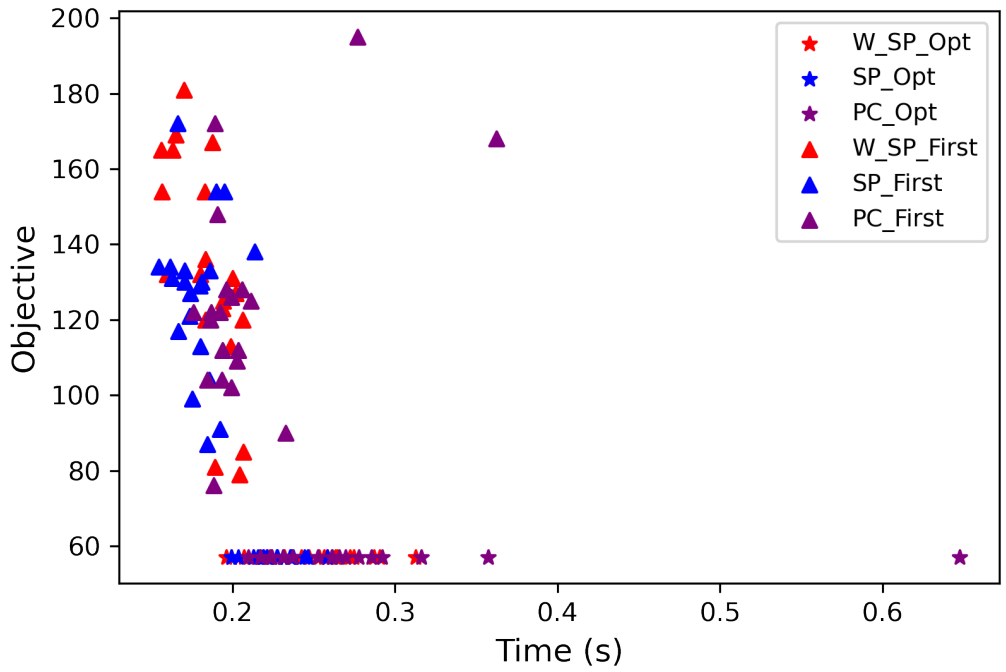


Figure B.2: Small layout, 5 vehicles, seed 2

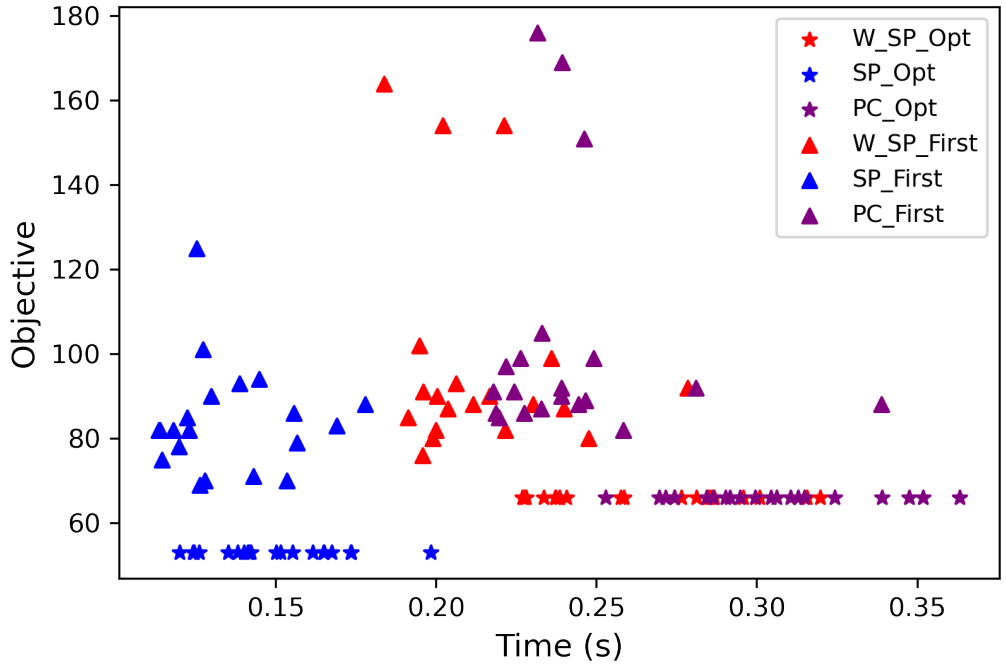


Figure B.3: Small layout, 5 vehicles, seed 3

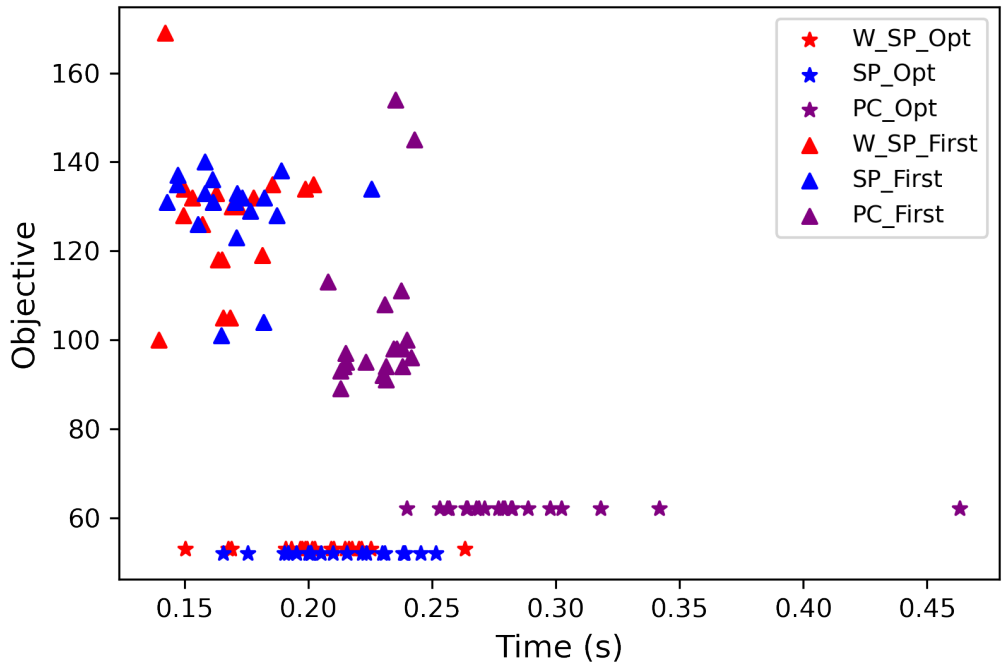


Figure B.4: Small layout, 5 vehicles, seed 4

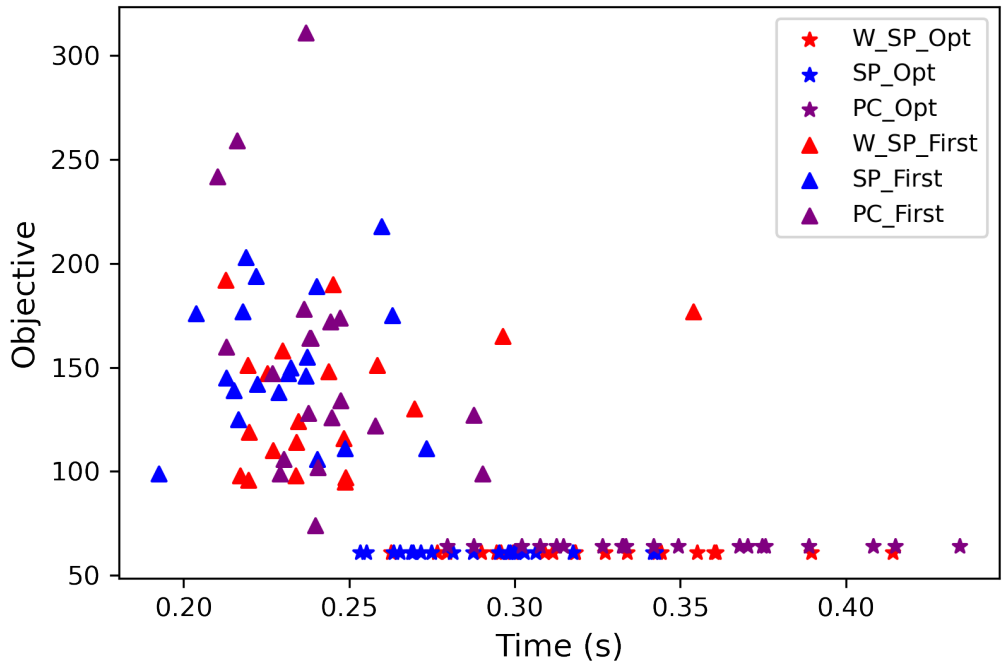


Figure B.5: Small layout, 5 vehicles, seed 5

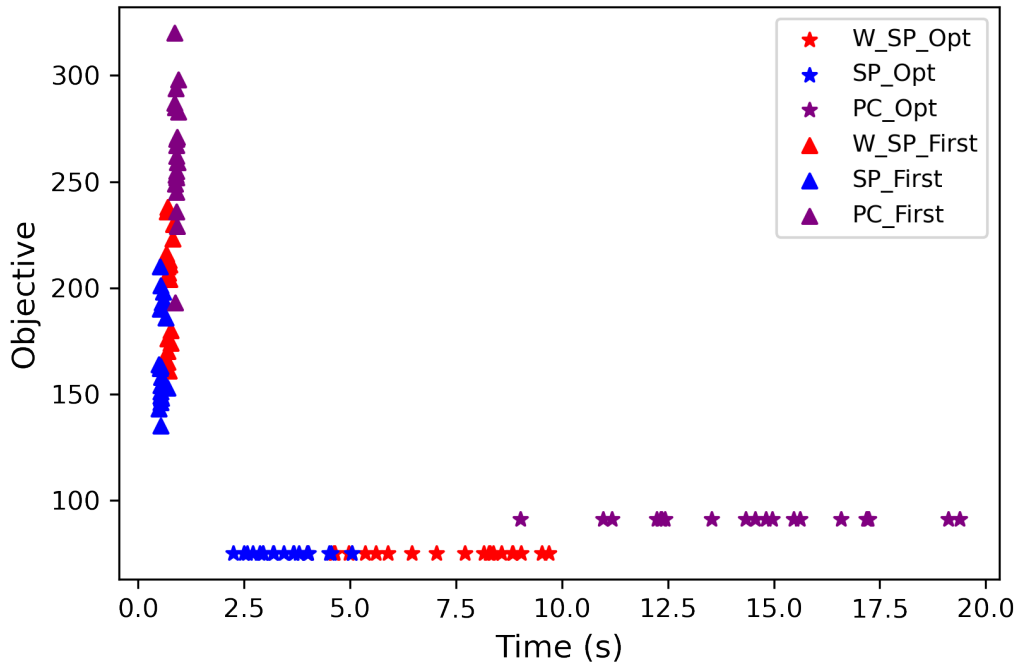


Figure B.6: Small layout, 10 vehicles, seed 1

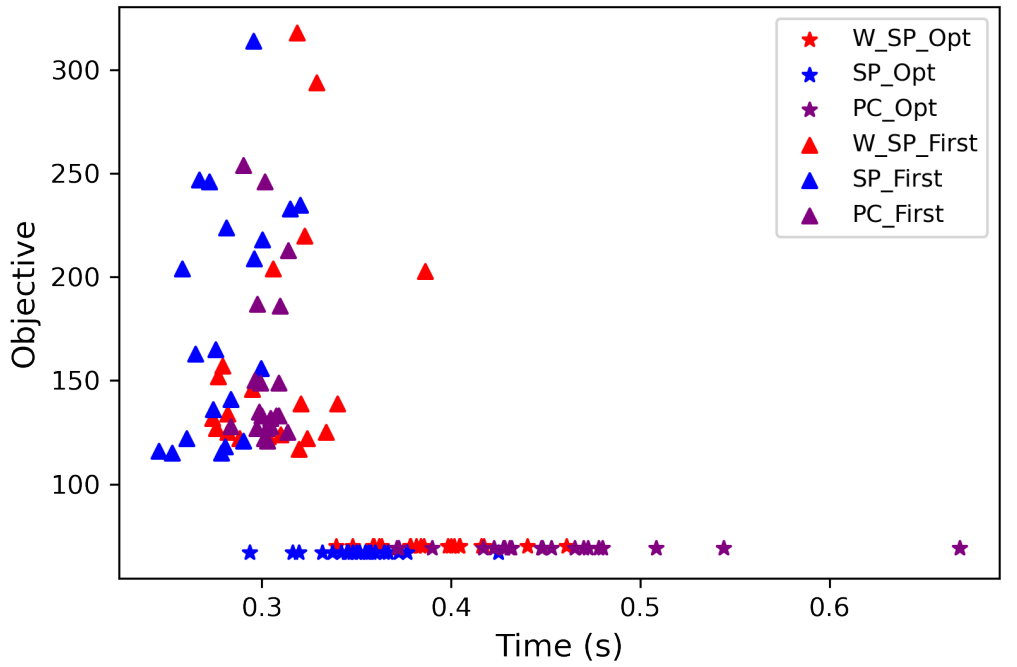


Figure B.7: Small layout, 10 vehicles, seed 2

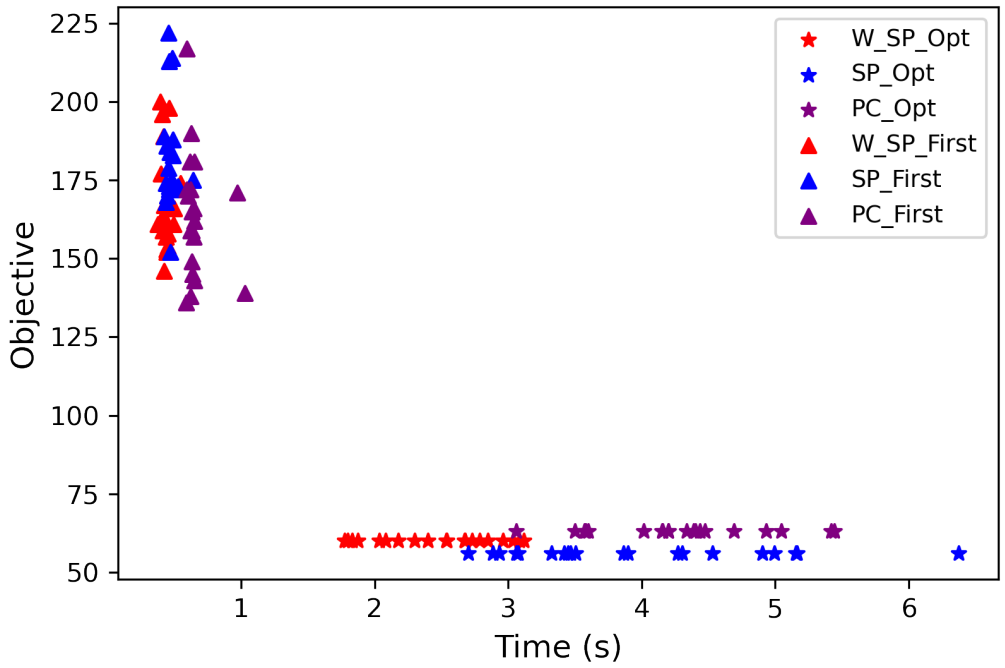


Figure B.8: Small layout, 10 vehicles, seed 3

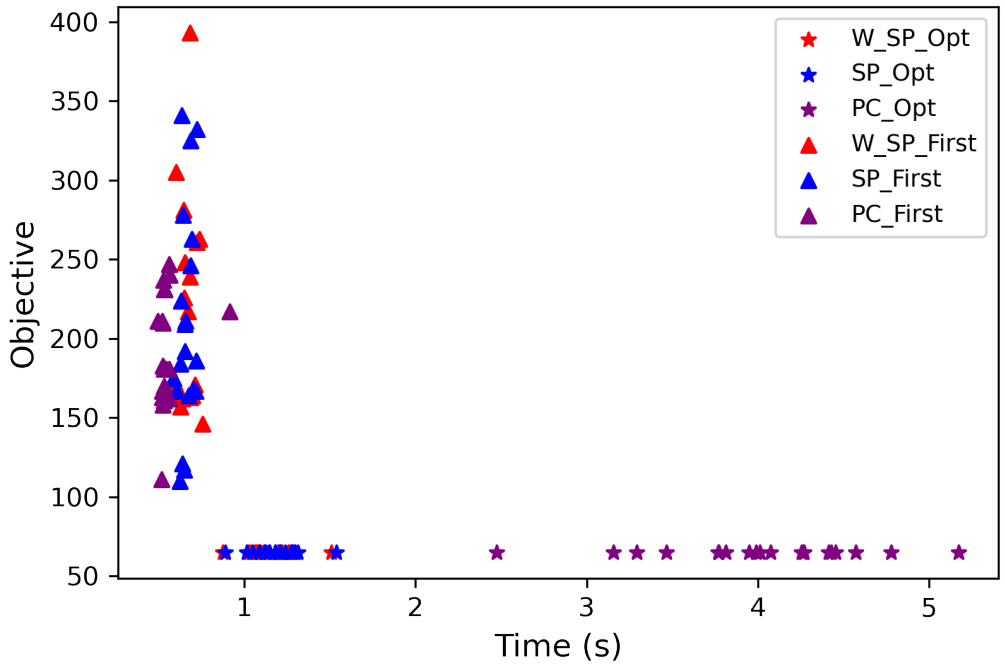


Figure B.9: Small layout, 10 vehicles, seed 4

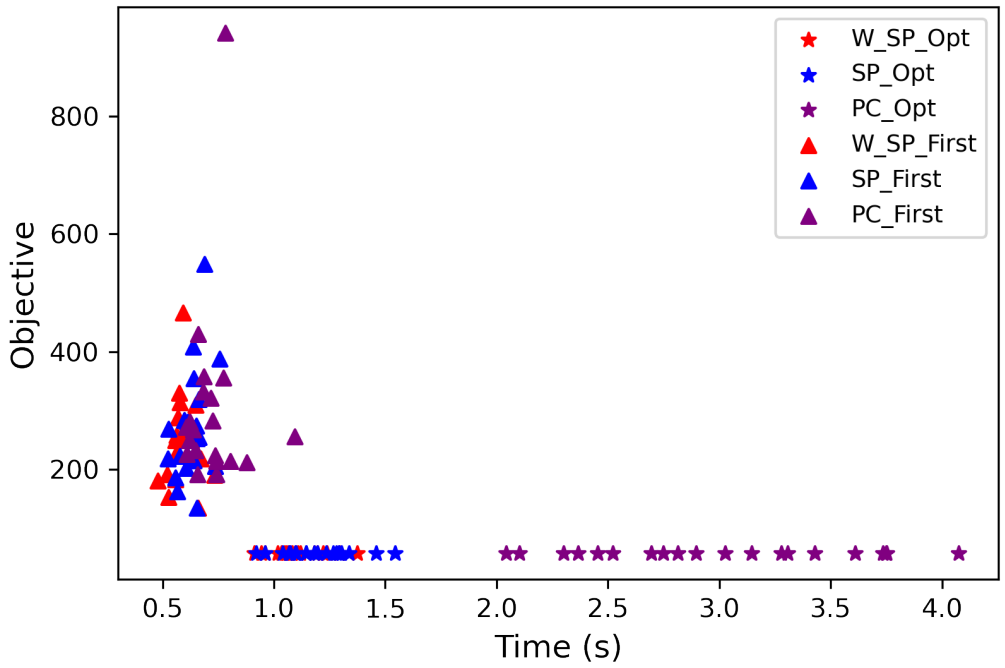


Figure B.10: Small layout, 10 vehicles, seed 5

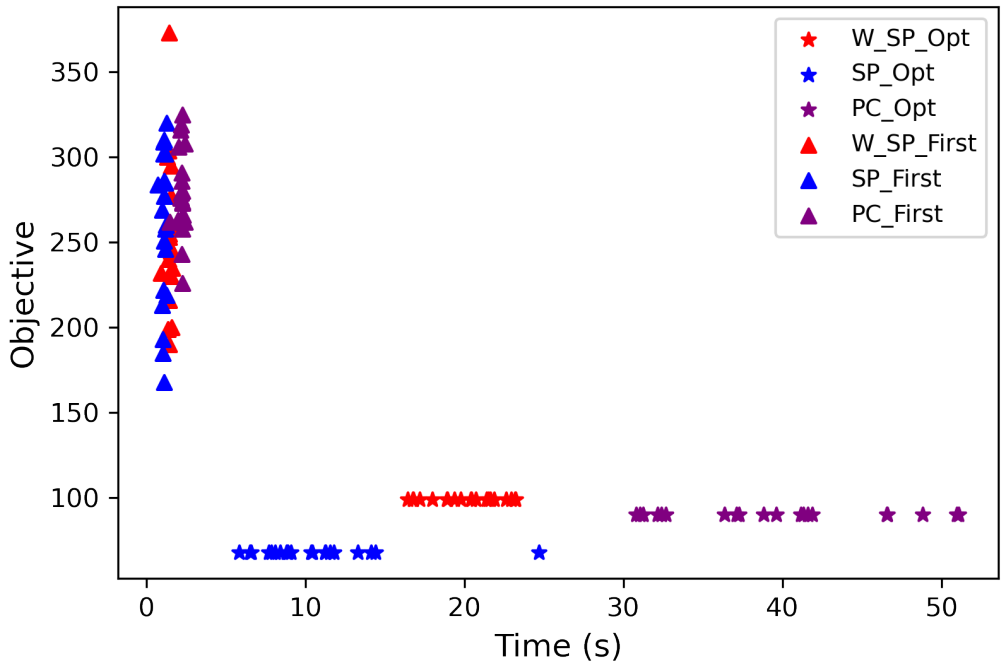


Figure B.11: Small layout, 15 vehicles, seed 1

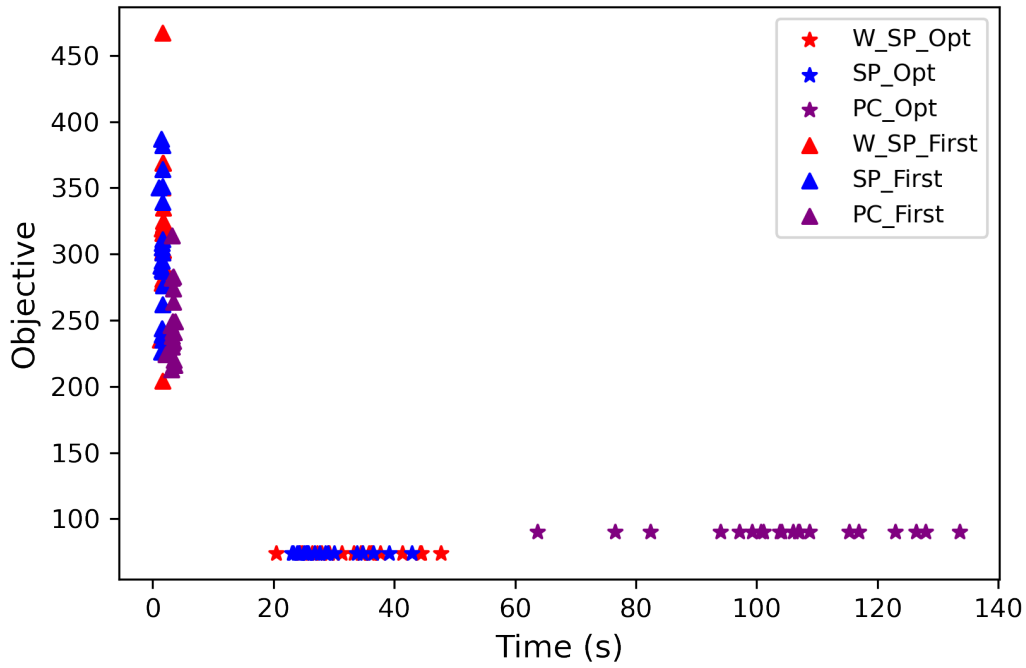


Figure B.12: Small layout, 15 vehicles, seed 2

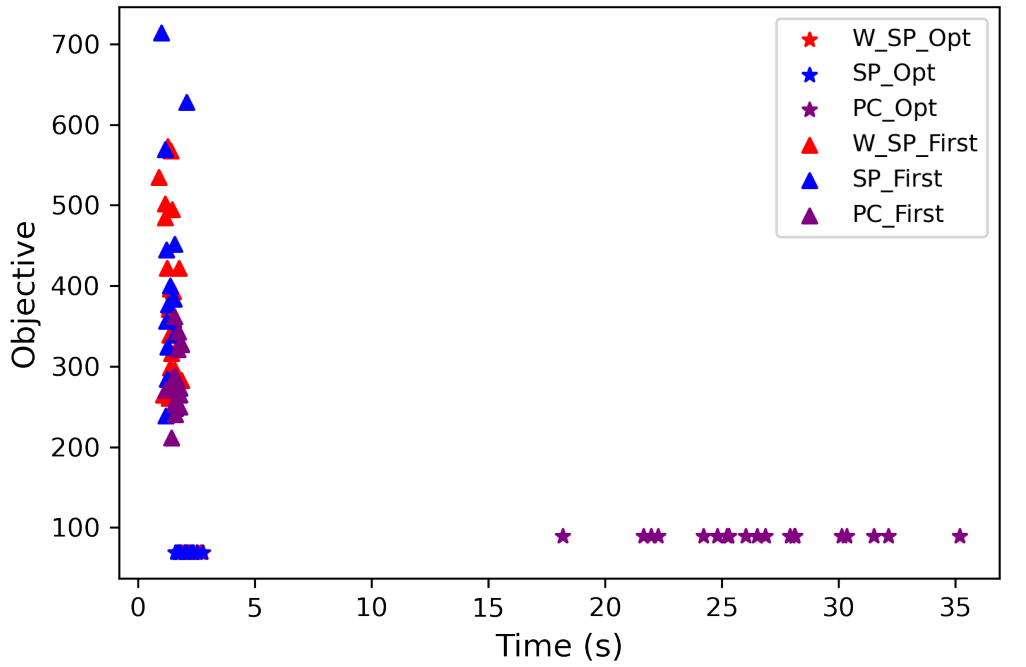


Figure B.13: Small layout, 15 vehicles, seed 3

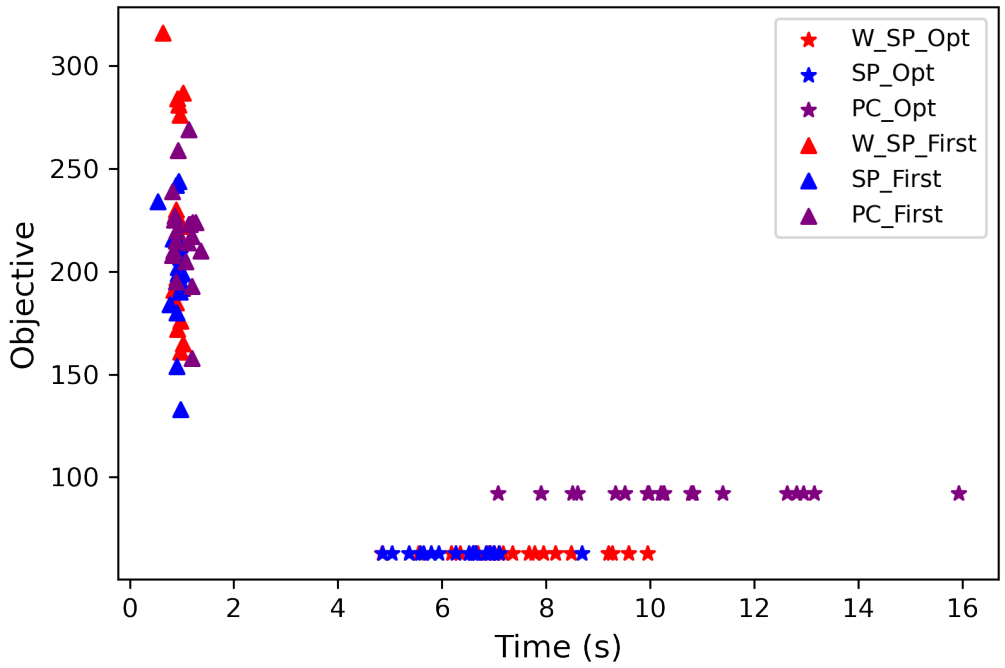


Figure B.14: Small layout, 15 vehicles, seed 4

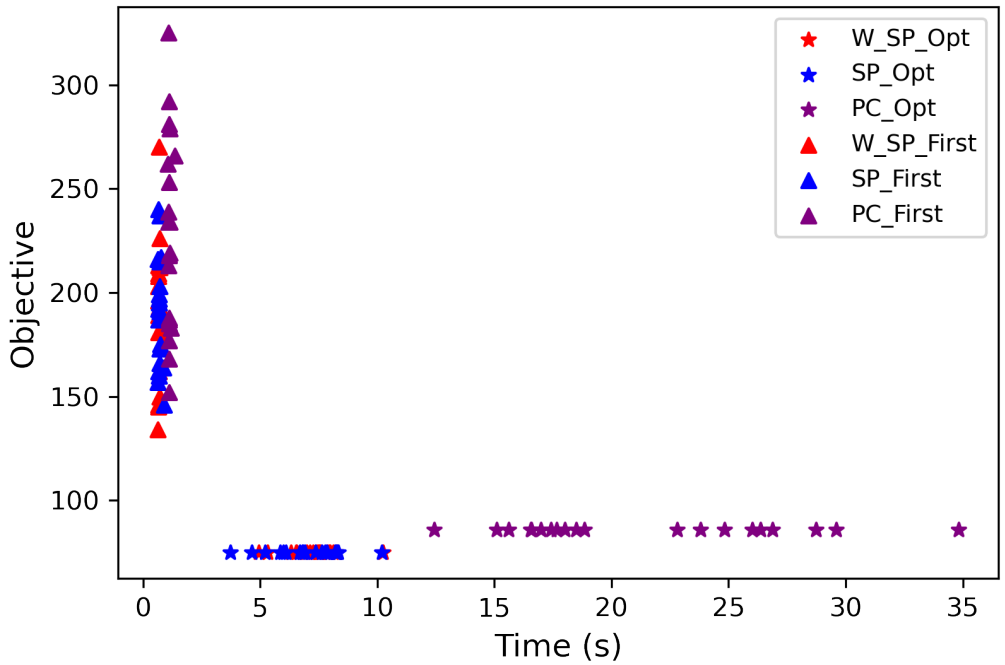


Figure B.15: Small layout, 15 vehicles, seed 5

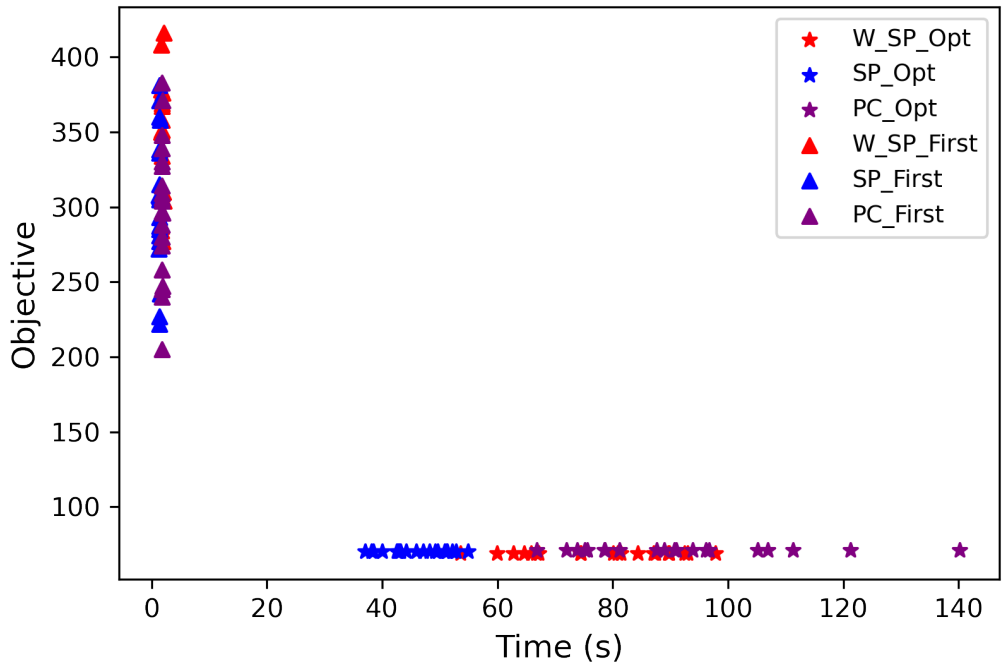


Figure B.16: Small layout, 20 vehicles, seed 1

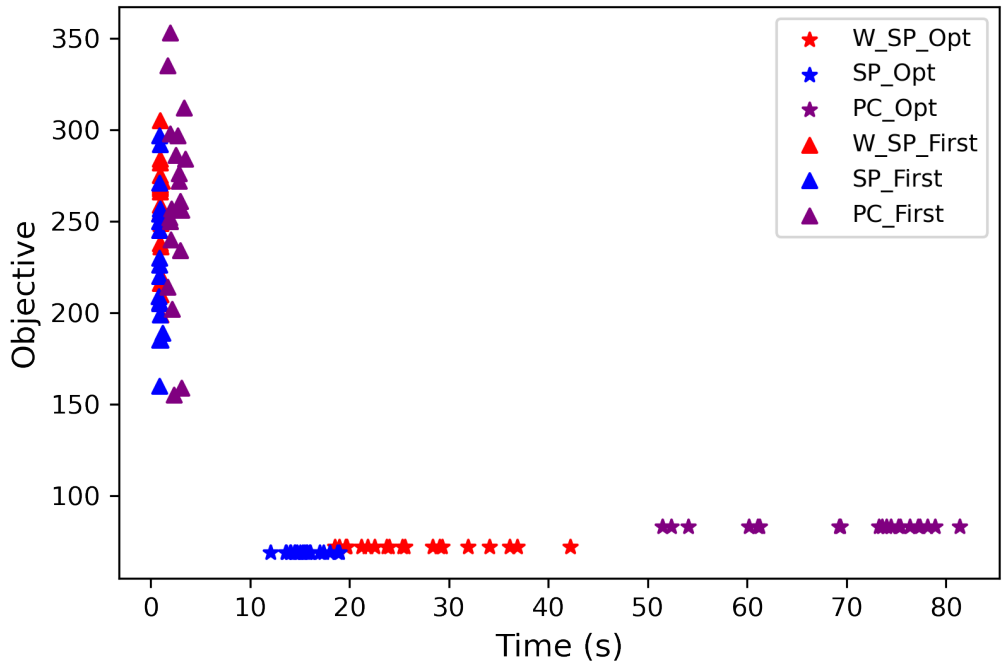


Figure B.17: Small layout, 20 vehicles, seed 2

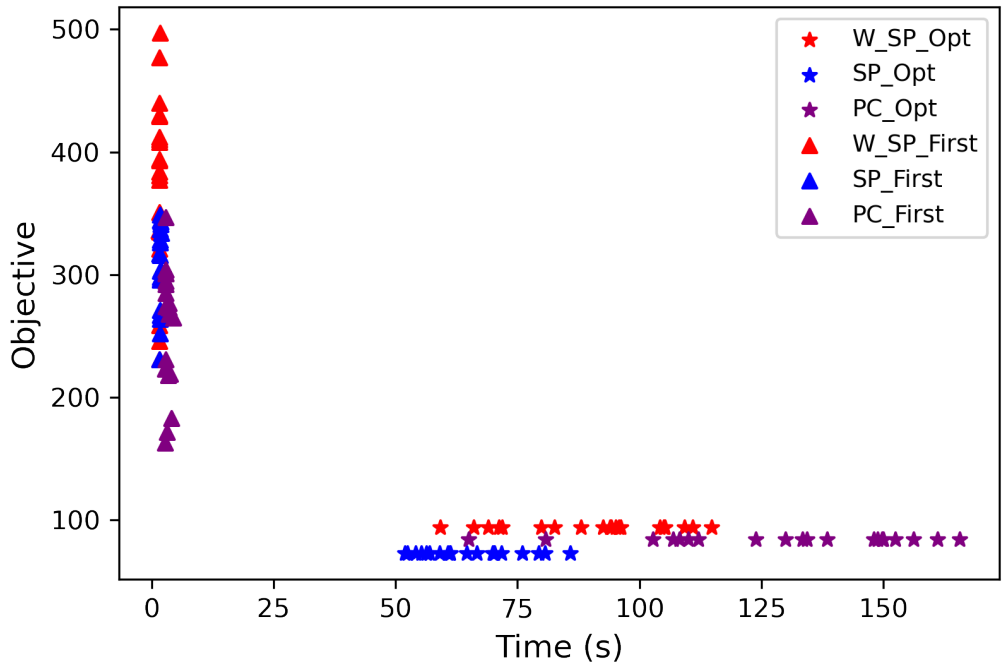


Figure B.18: Small layout, 20 vehicles, seed 3

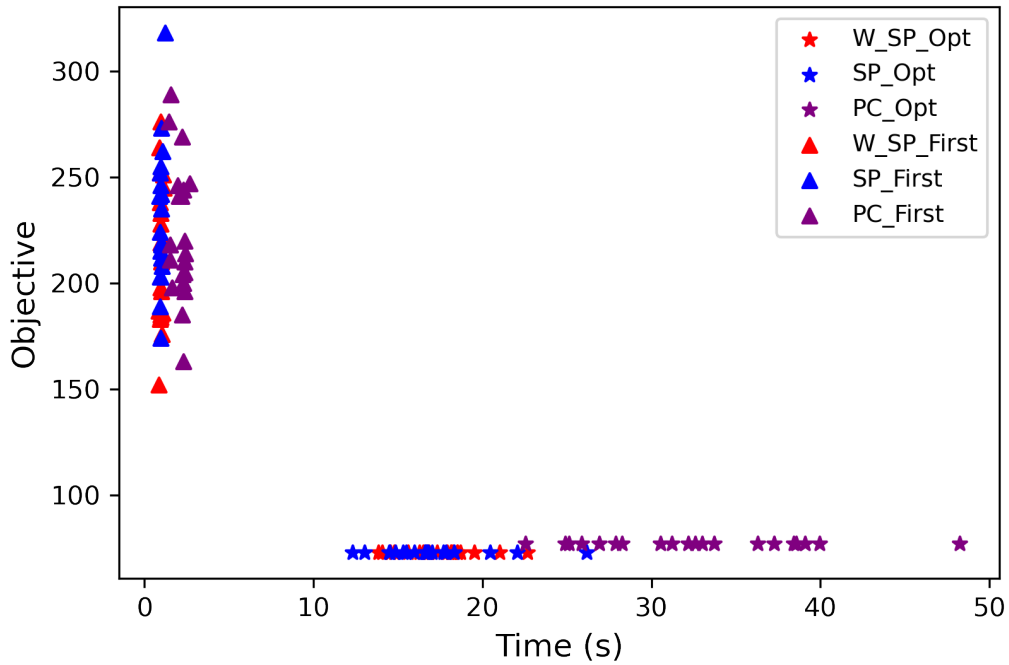


Figure B.19: Small layout, 20 vehicles, seed 4

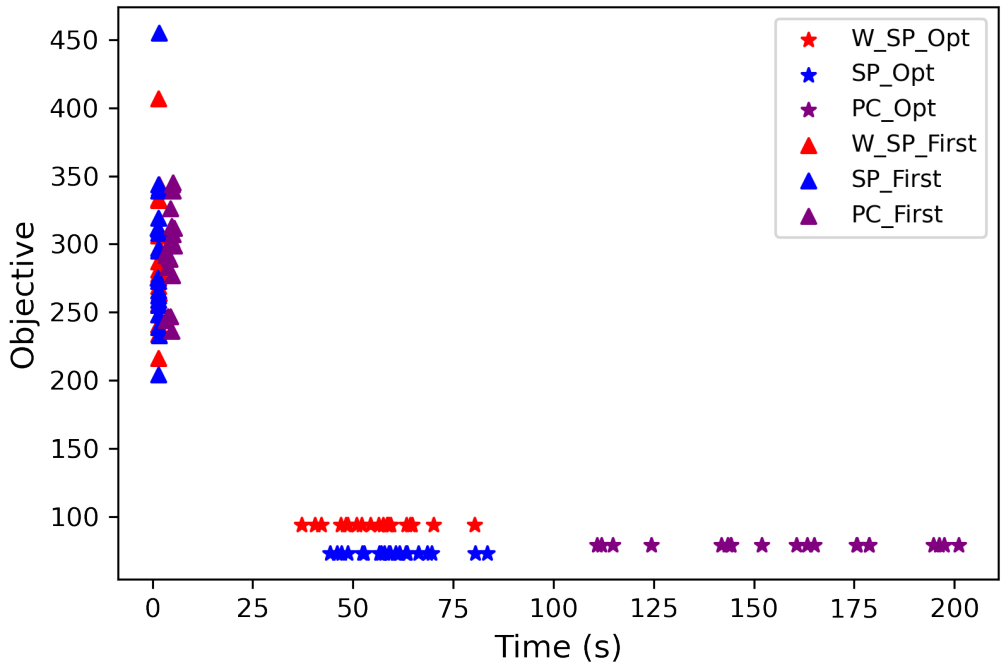


Figure B.20: Small layout, 20 vehicles, seed 5

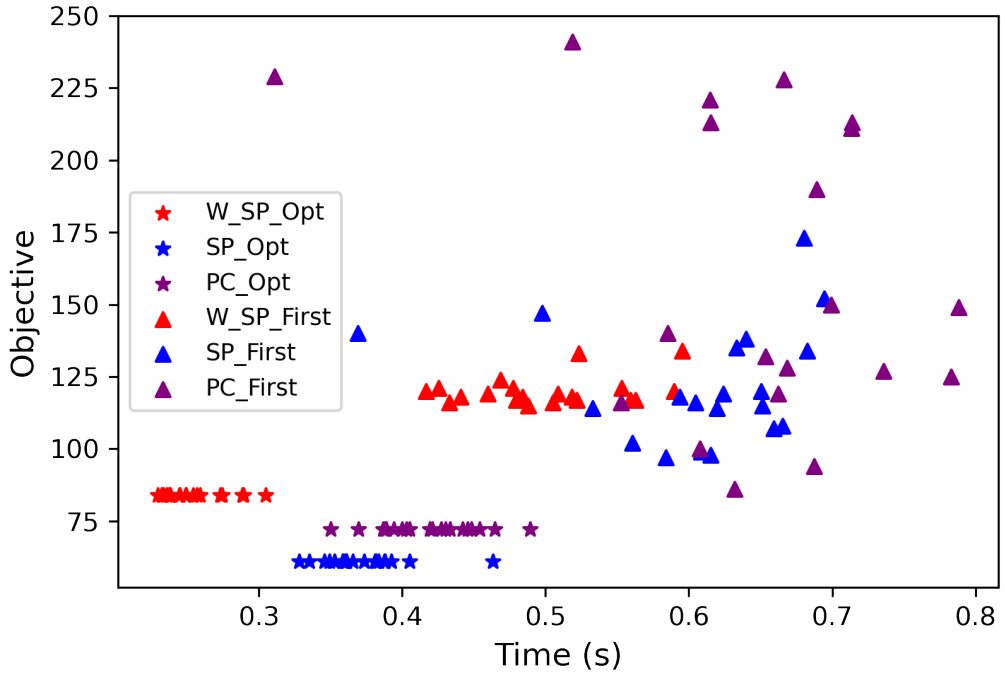


Figure B.21: Medium layout, 5 vehicles, seed 1

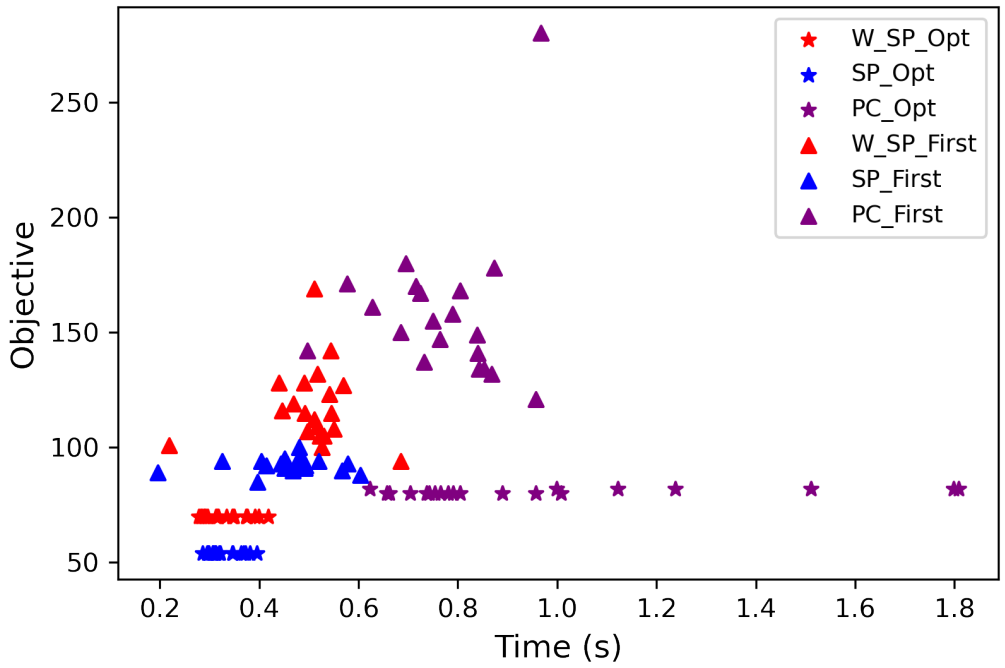


Figure B.22: Medium layout, 5 vehicles, seed 2

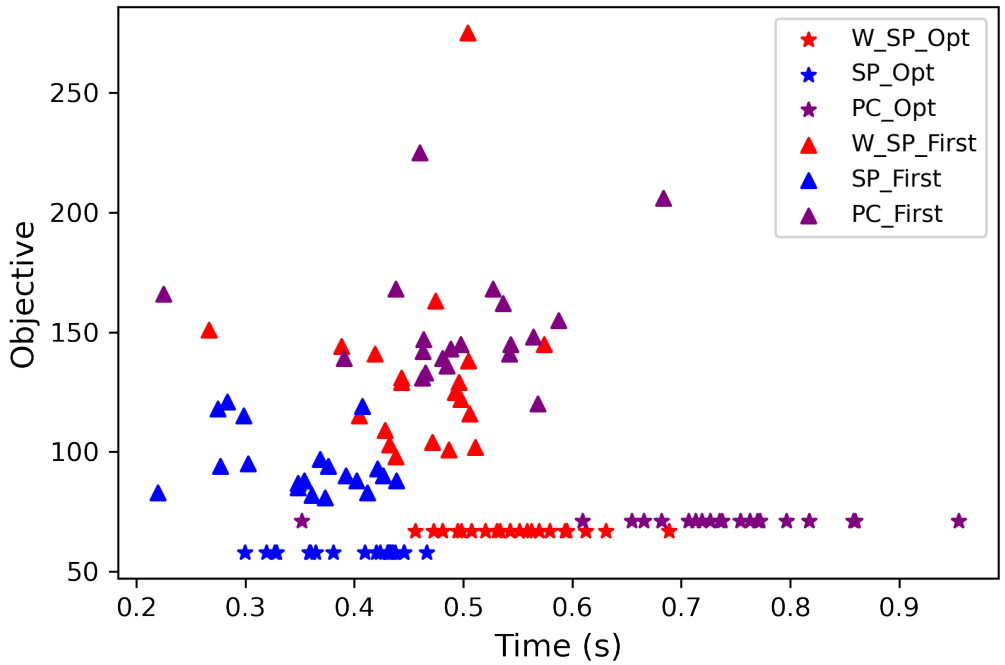


Figure B.23: Medium layout, 5 vehicles, seed 3

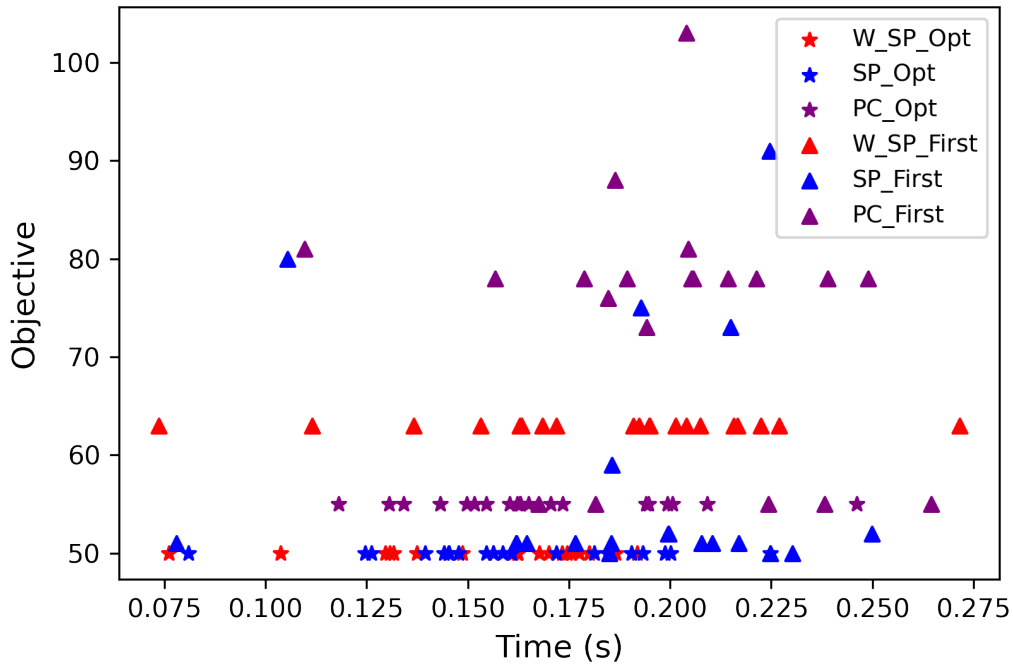


Figure B.24: Medium layout, 5 vehicles, seed 4

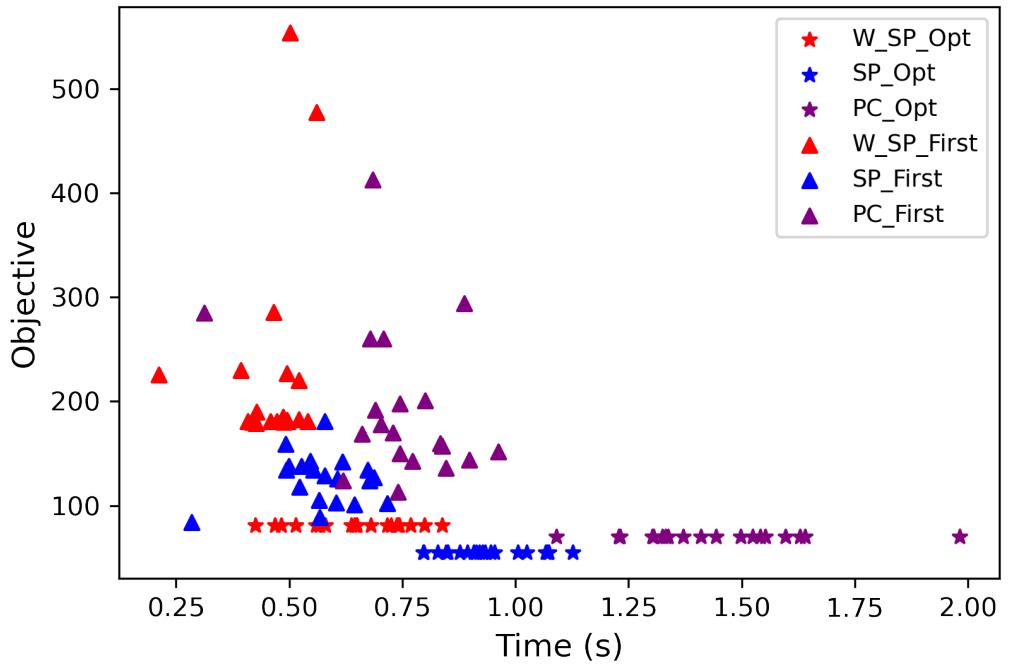


Figure B.25: Medium layout, 5 vehicles, seed 5

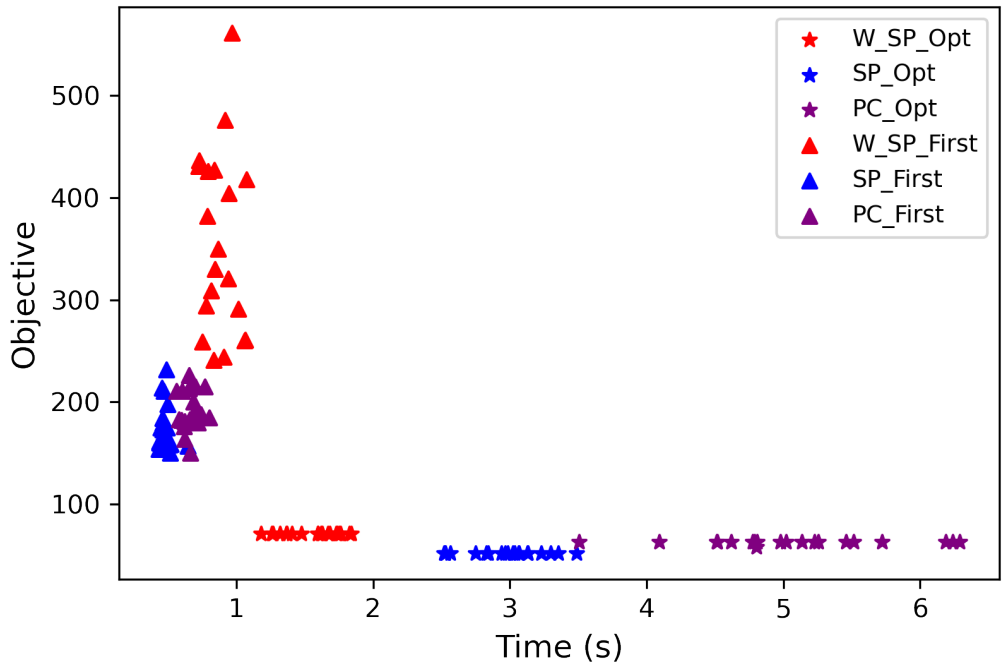


Figure B.26: Medium layout, 10 vehicles, seed 1

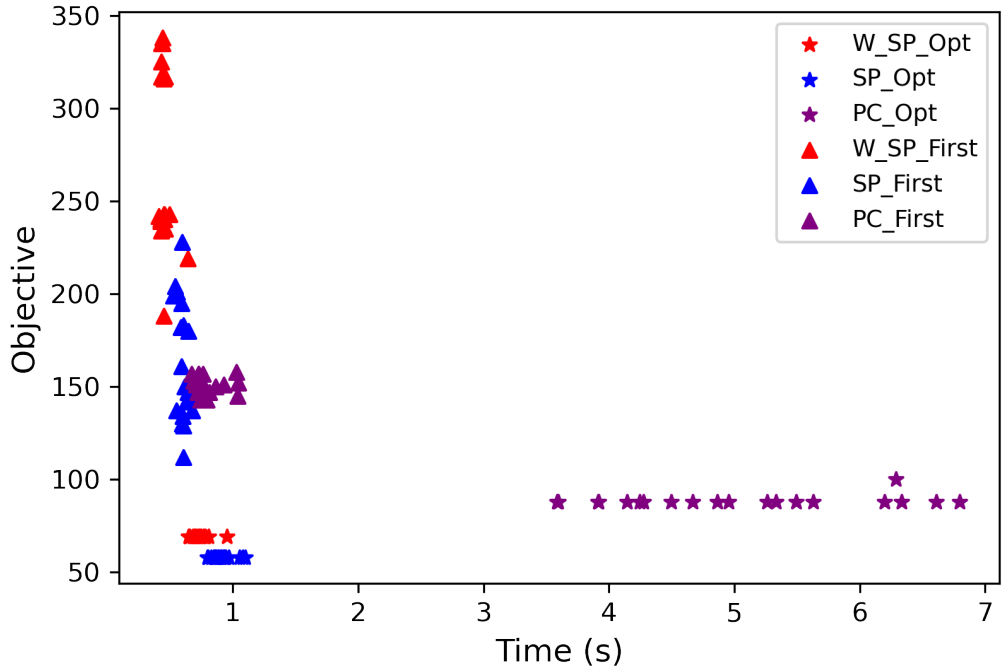


Figure B.27: Medium layout, 10 vehicles, seed 2

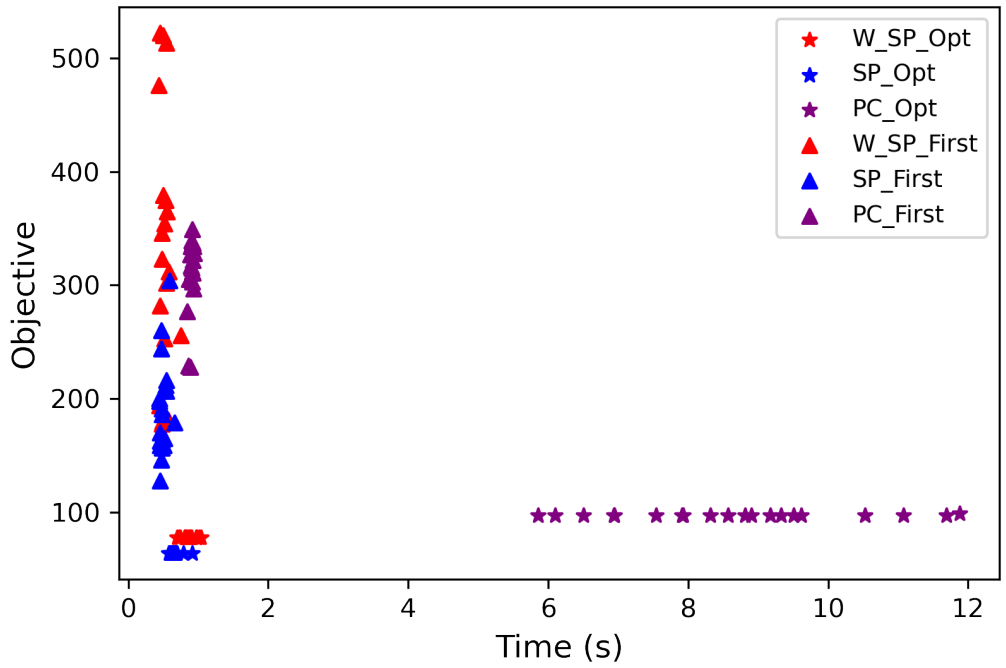


Figure B.28: Medium layout, 10 vehicles, seed 3

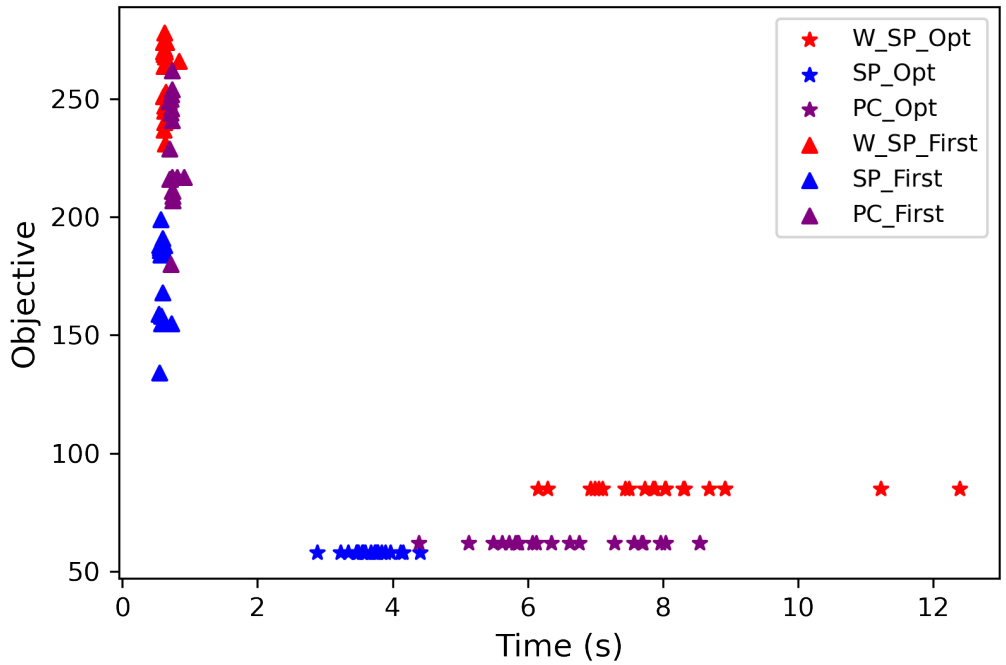


Figure B.29: Medium layout, 10 vehicles, seed 4

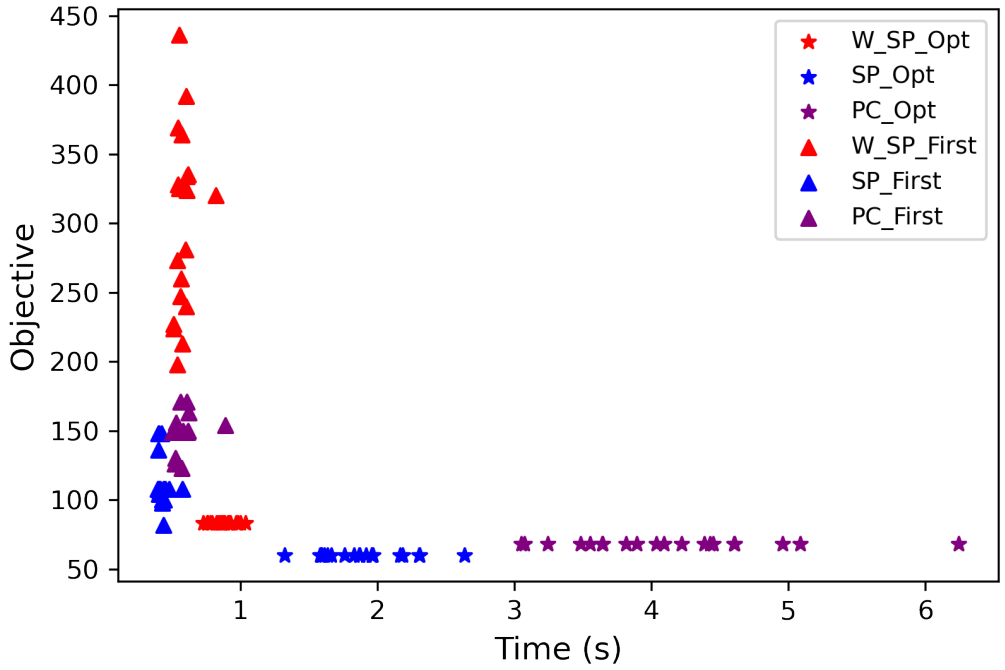


Figure B.30: Medium layout, 10 vehicles, seed 5

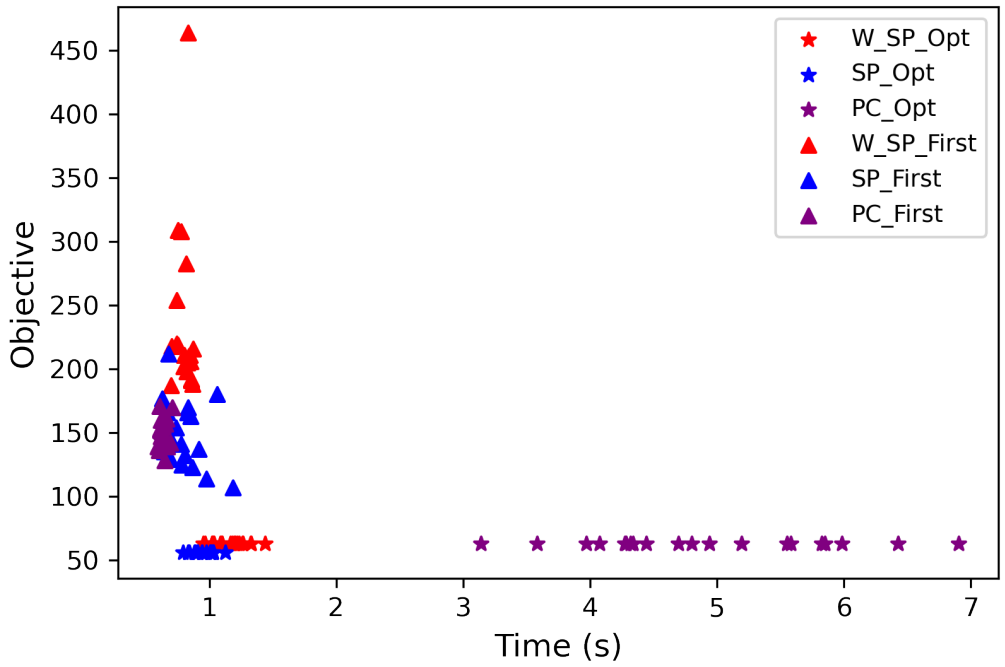


Figure B.31: Medium layout, 15 vehicles, seed 1

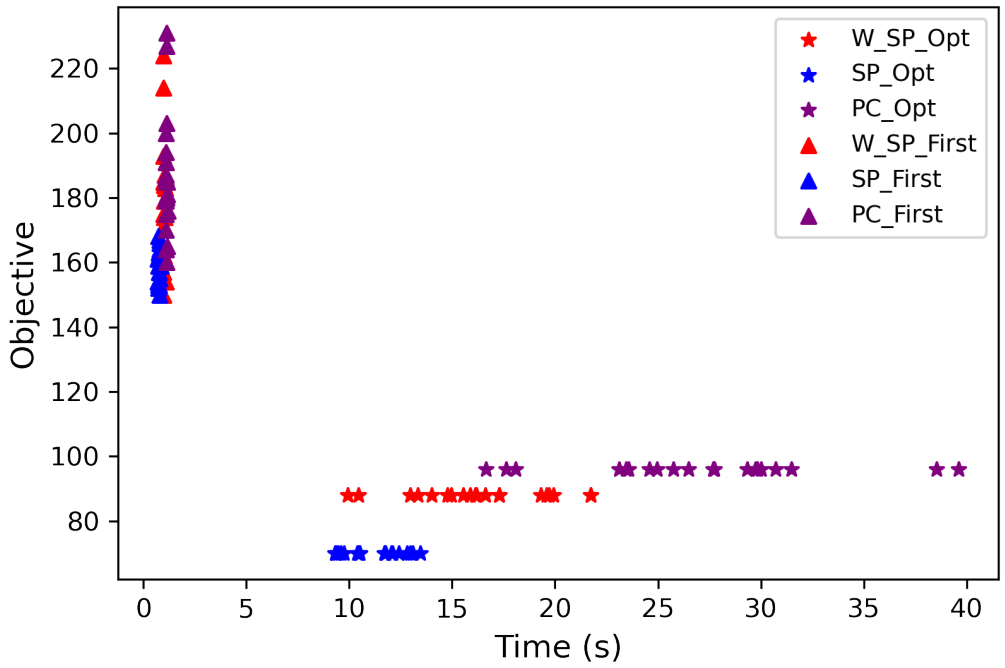


Figure B.32: Medium layout, 15 vehicles, seed 2

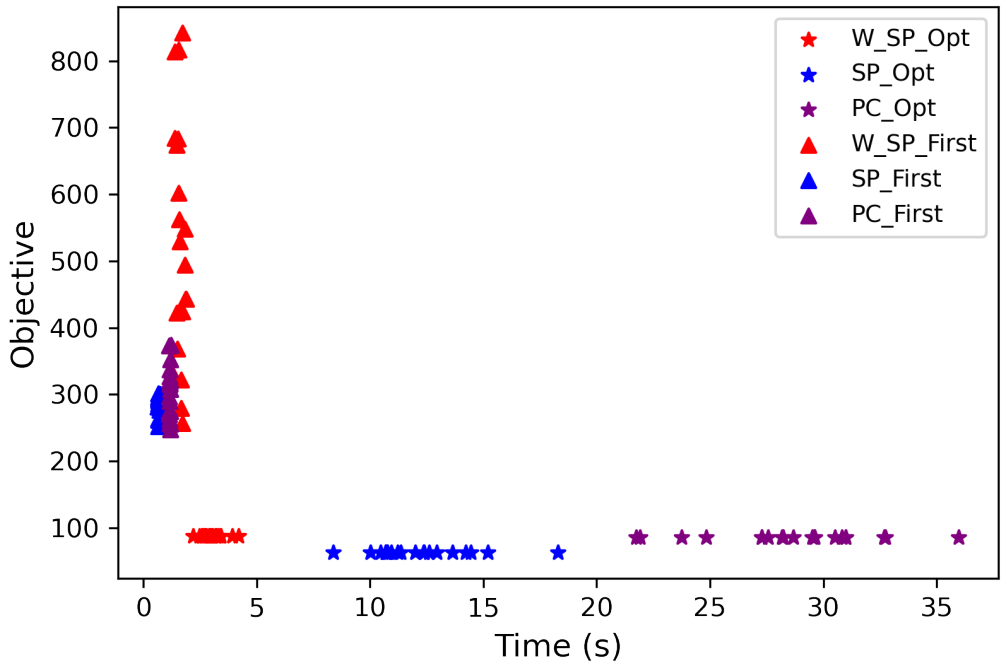


Figure B.33: Medium layout, 15 vehicles, seed 3

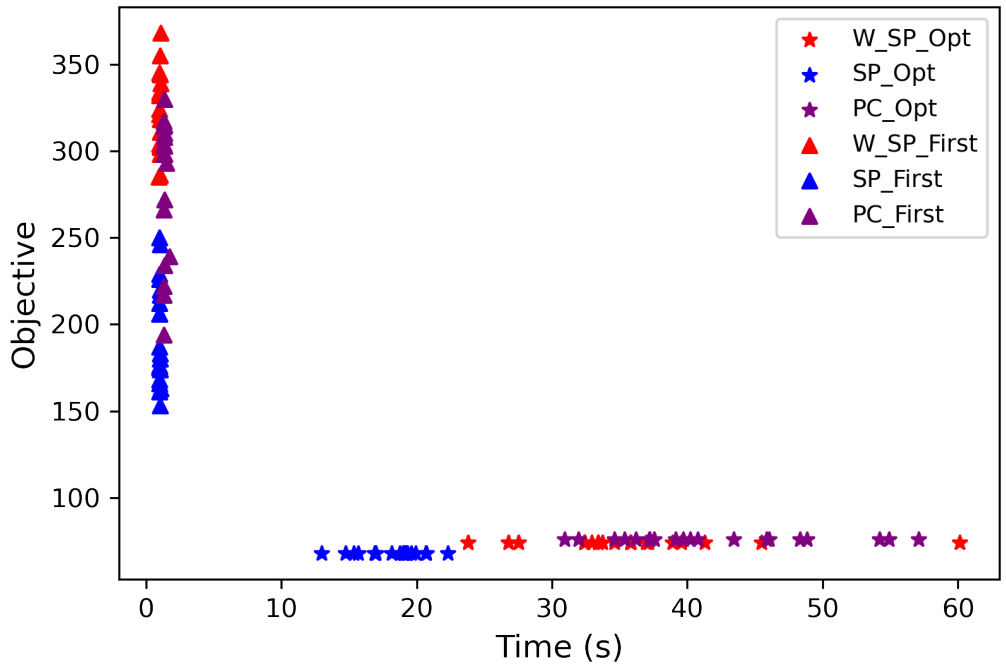


Figure B.34: Medium layout, 15 vehicles, seed 4

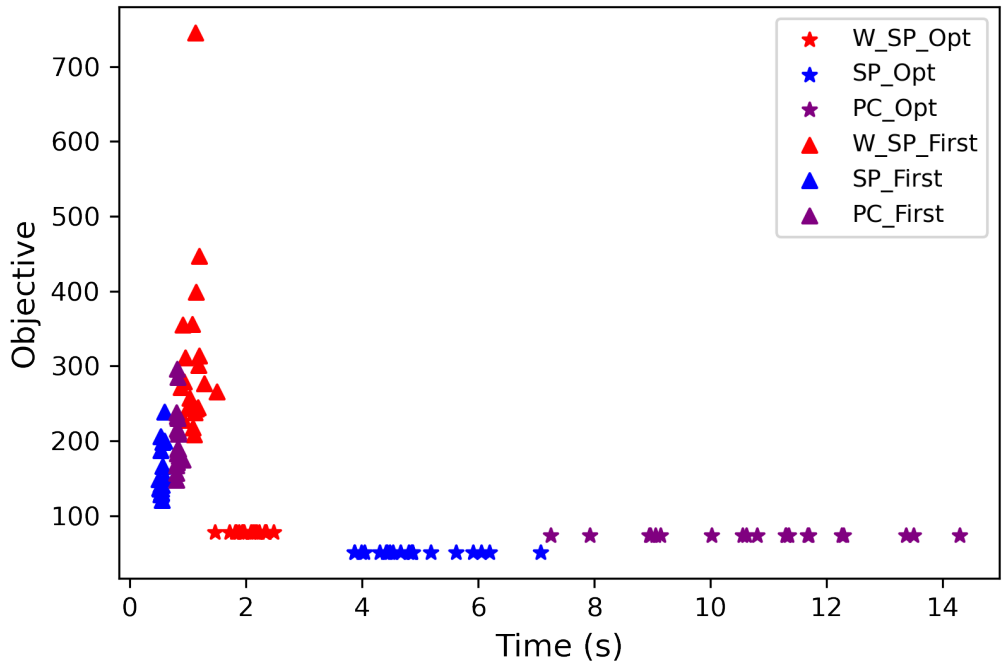


Figure B.35: Medium layout, 15 vehicles, seed 5

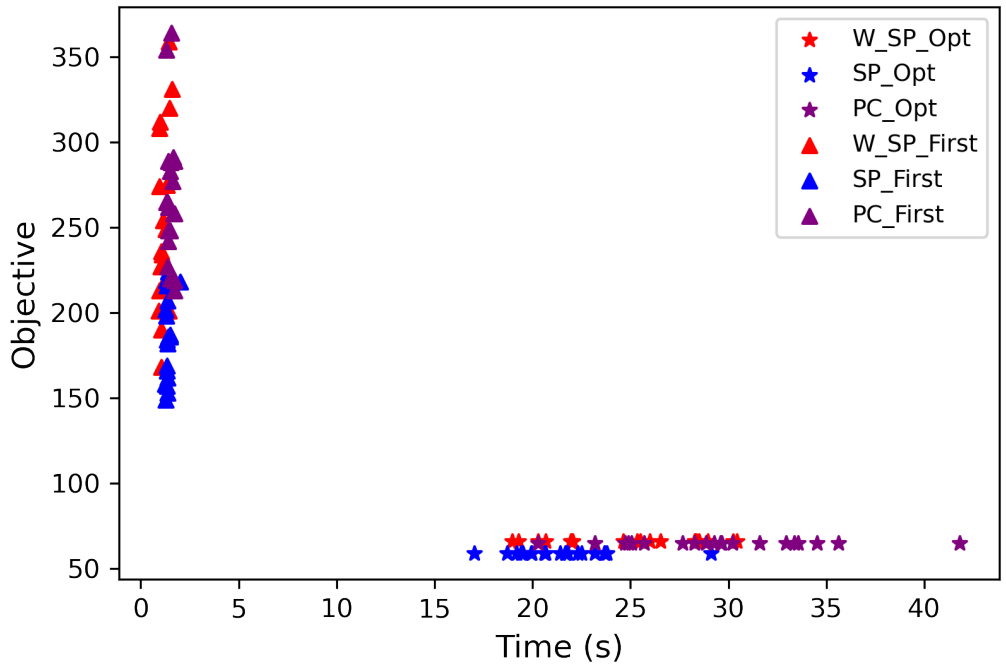


Figure B.36: Medium layout, 20 vehicles, seed 1

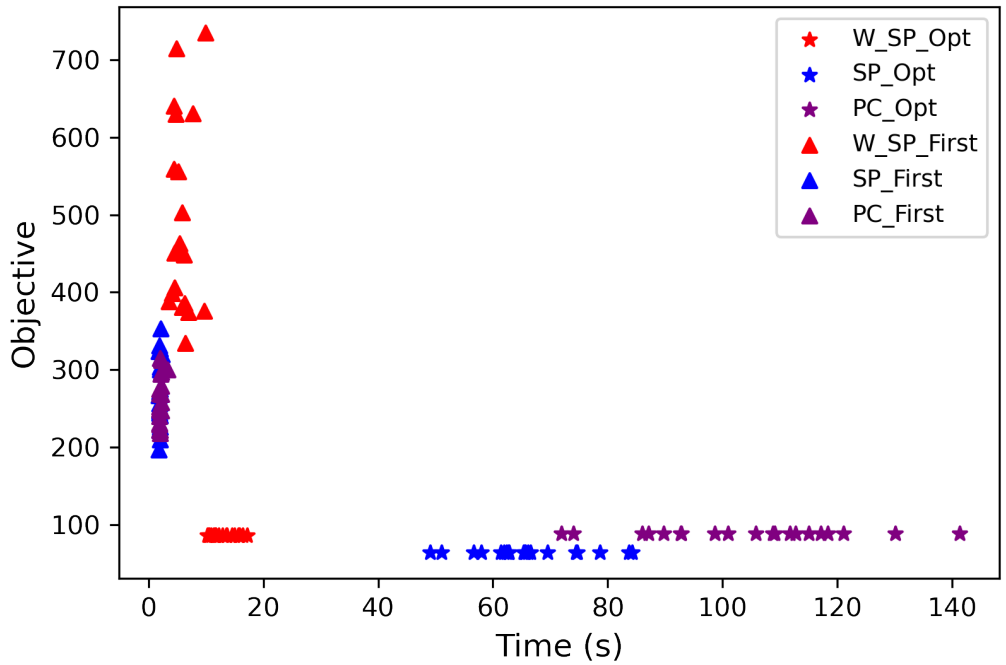


Figure B.37: Medium layout, 20 vehicles, seed 2

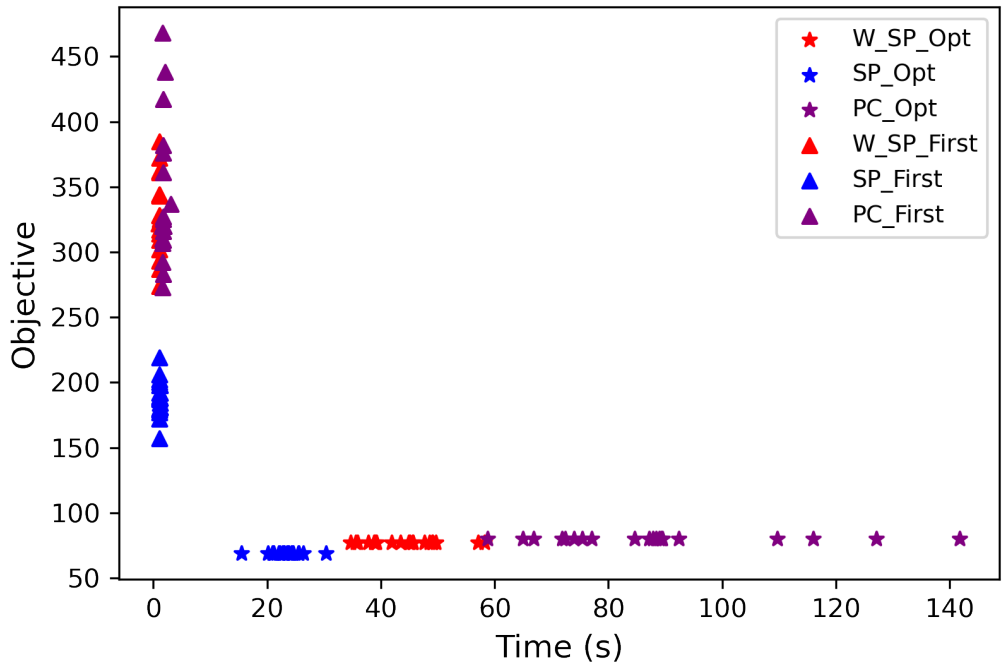


Figure B.38: Medium layout, 20 vehicles, seed 3

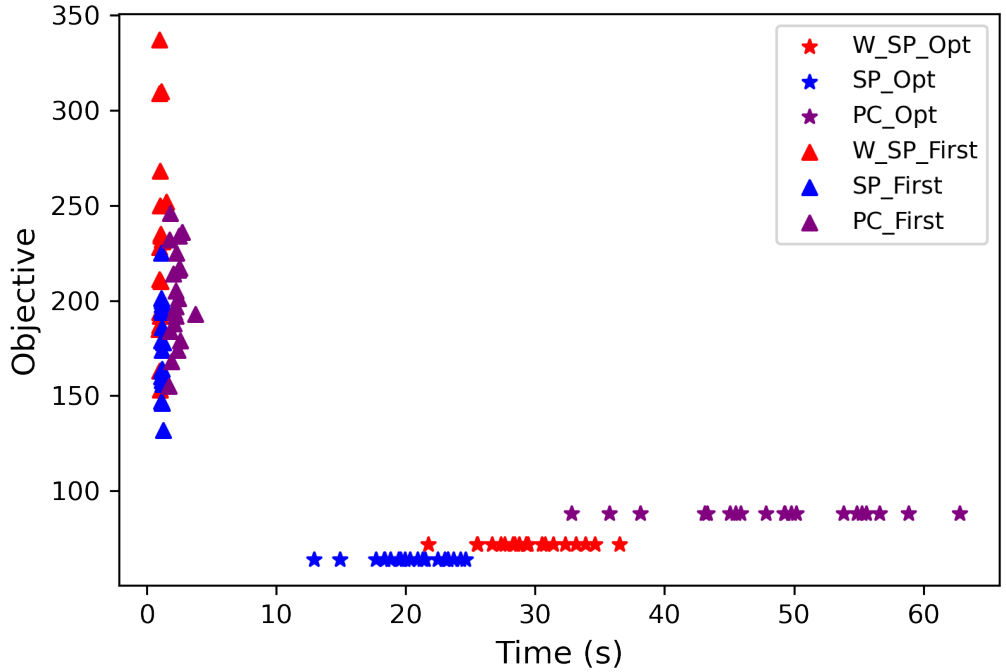


Figure B.39: Medium layout, 20 vehicles, seed 4

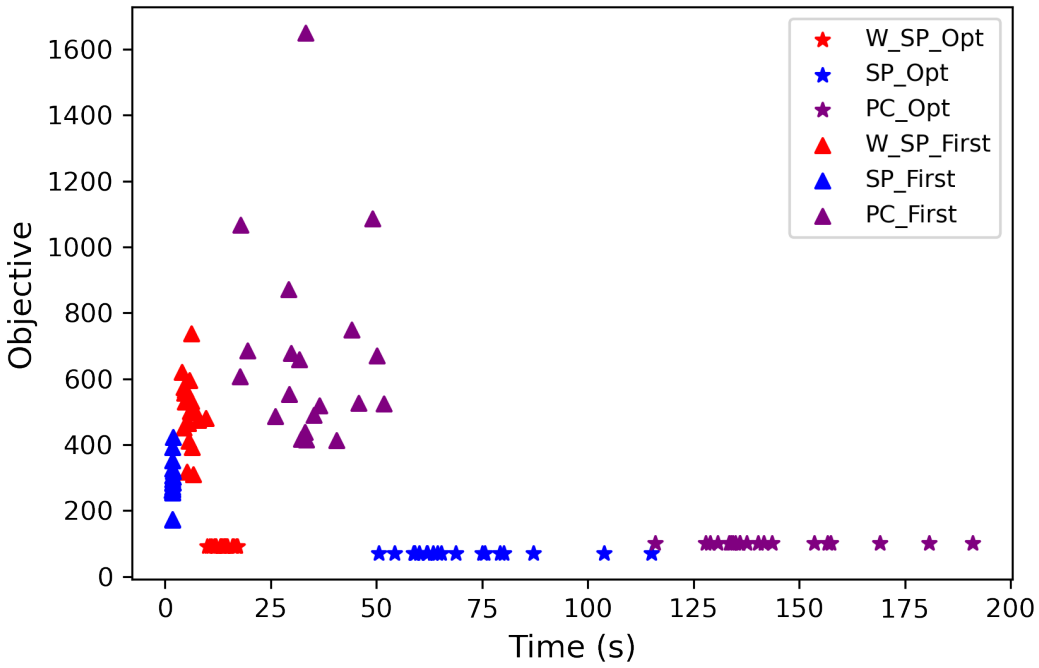


Figure B.40: Medium layout, 20 vehicles, seed 5

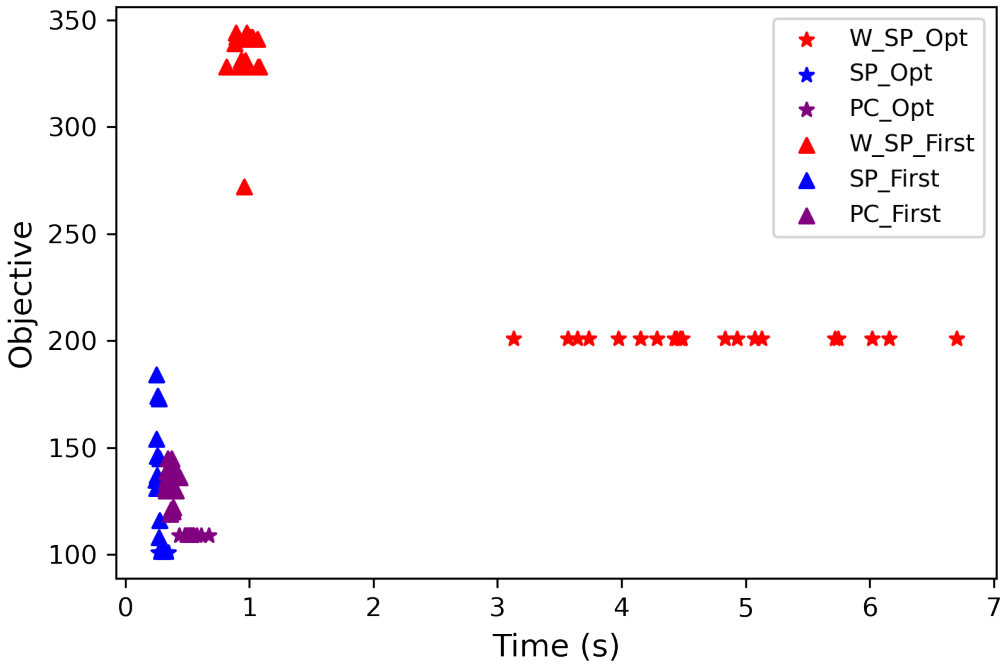


Figure B.41: Large layout, 5 vehicles, seed 1

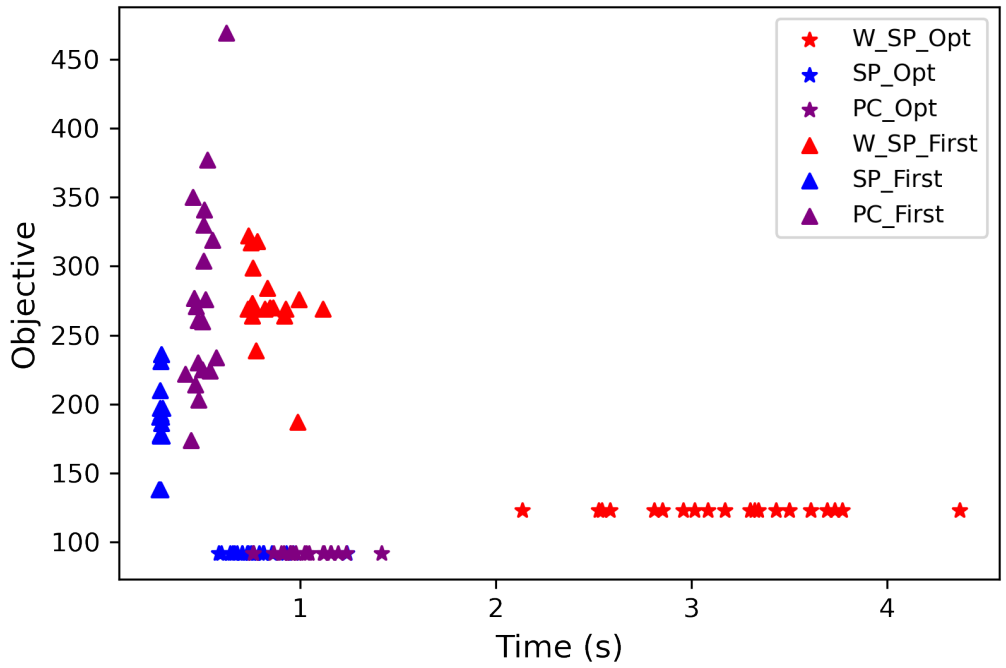


Figure B.42: Large layout, 5 vehicles, seed 2

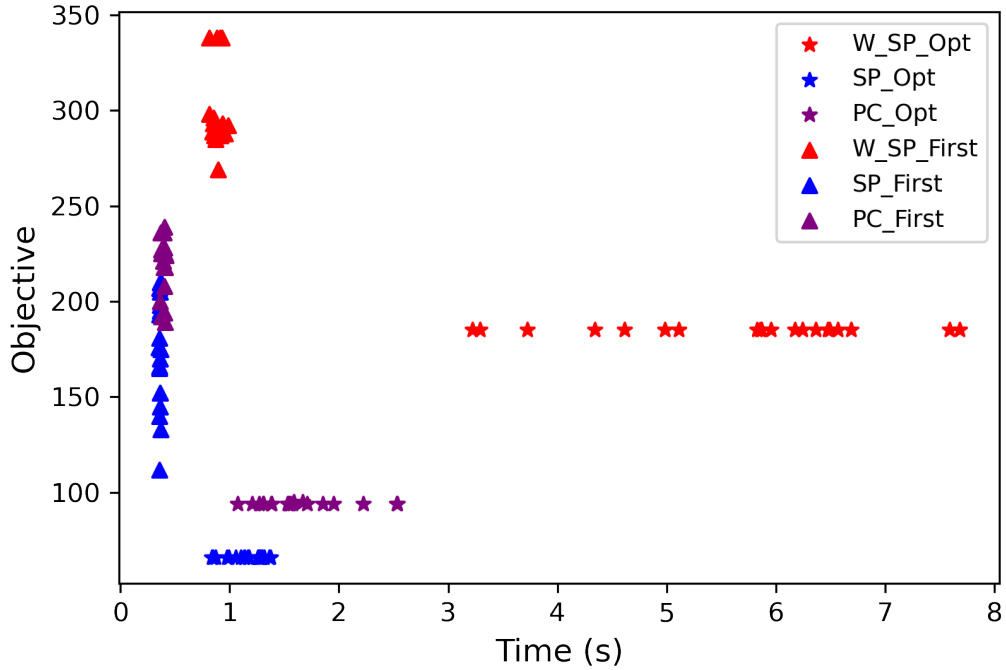


Figure B.43: Large layout, 5 vehicles, seed 3

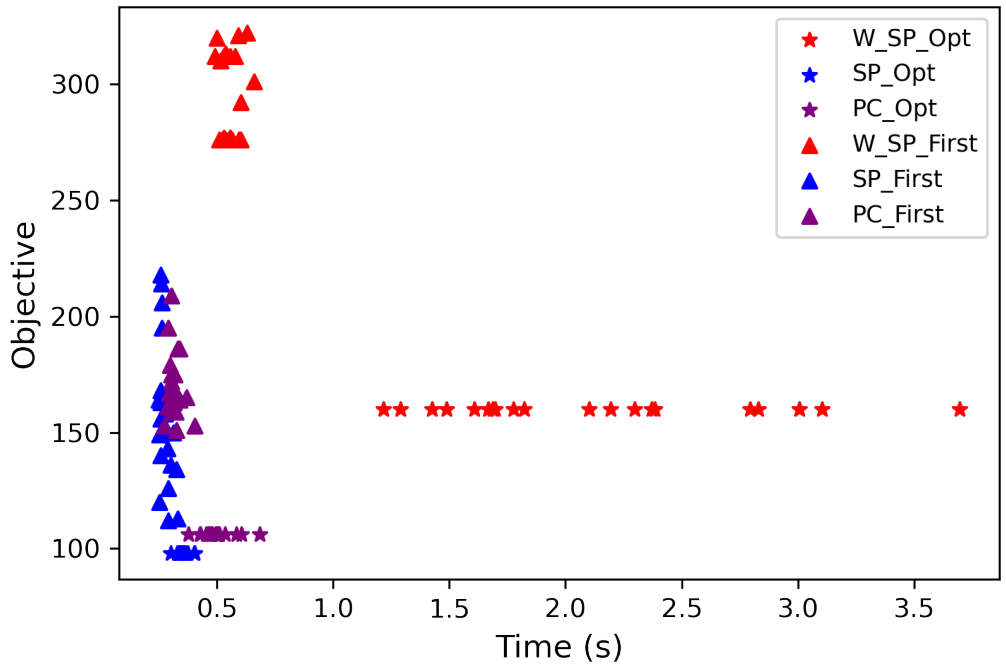


Figure B.44: Large layout, 5 vehicles, seed 4

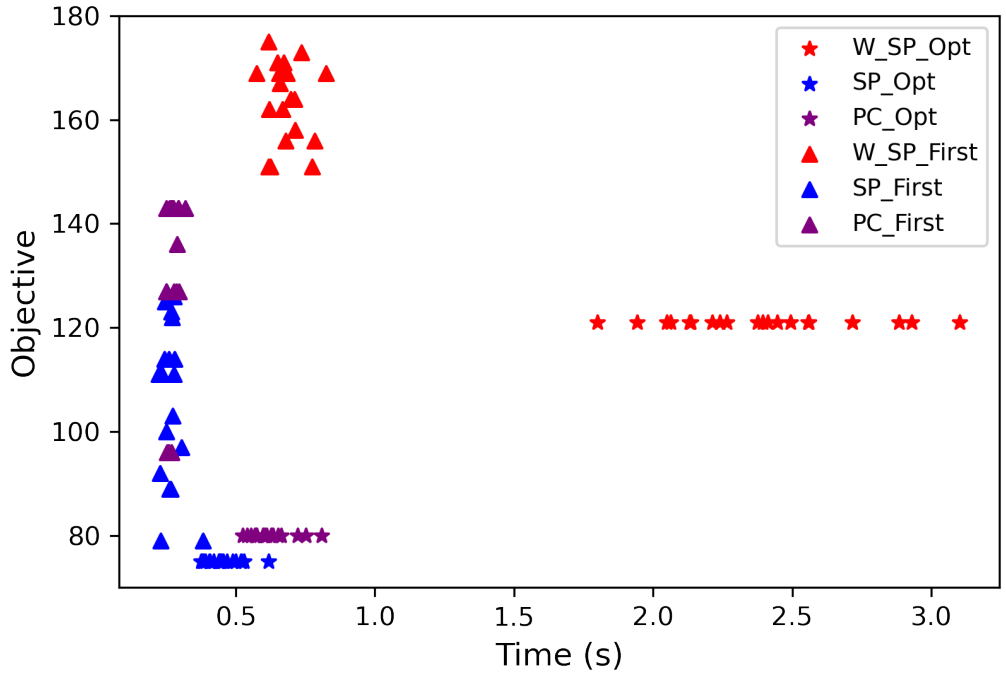


Figure B.45: Large layout, 5 vehicles, seed 5

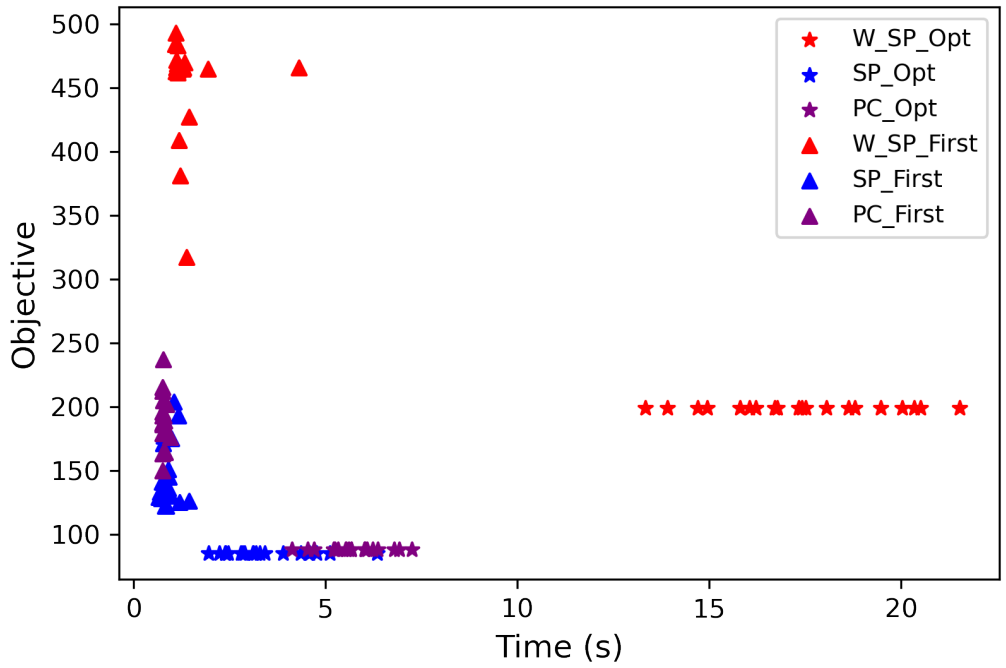


Figure B.46: Large layout, 10 vehicles, seed 1

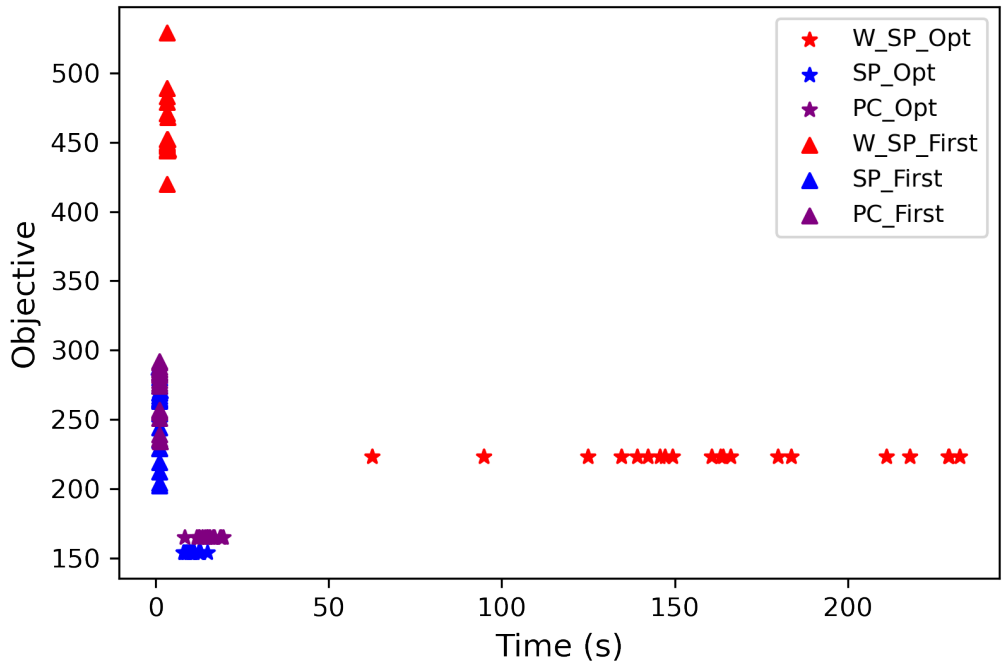


Figure B.47: Large layout, 10 vehicles, seed 2

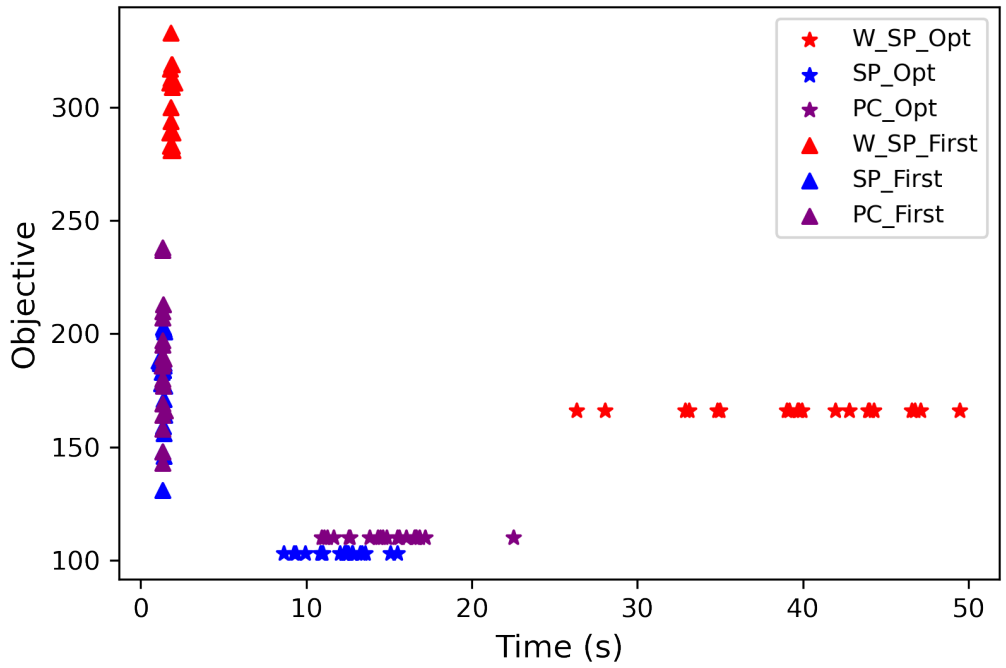


Figure B.48: Large layout, 10 vehicles, seed 3

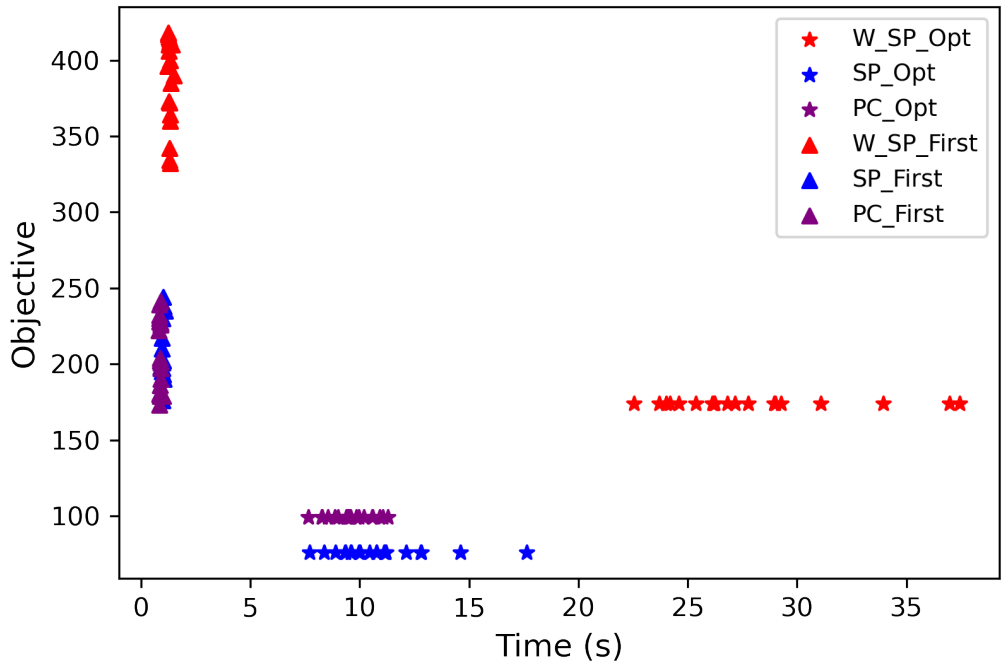


Figure B.49: Large layout, 10 vehicles, seed 4

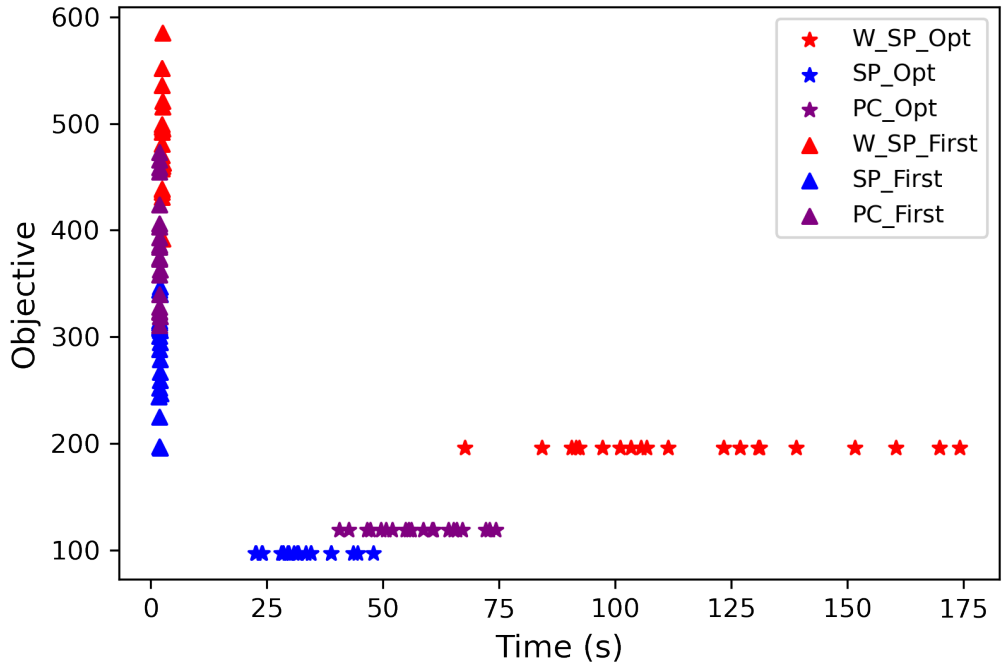


Figure B.50: Large layout, 10 vehicles, seed 5

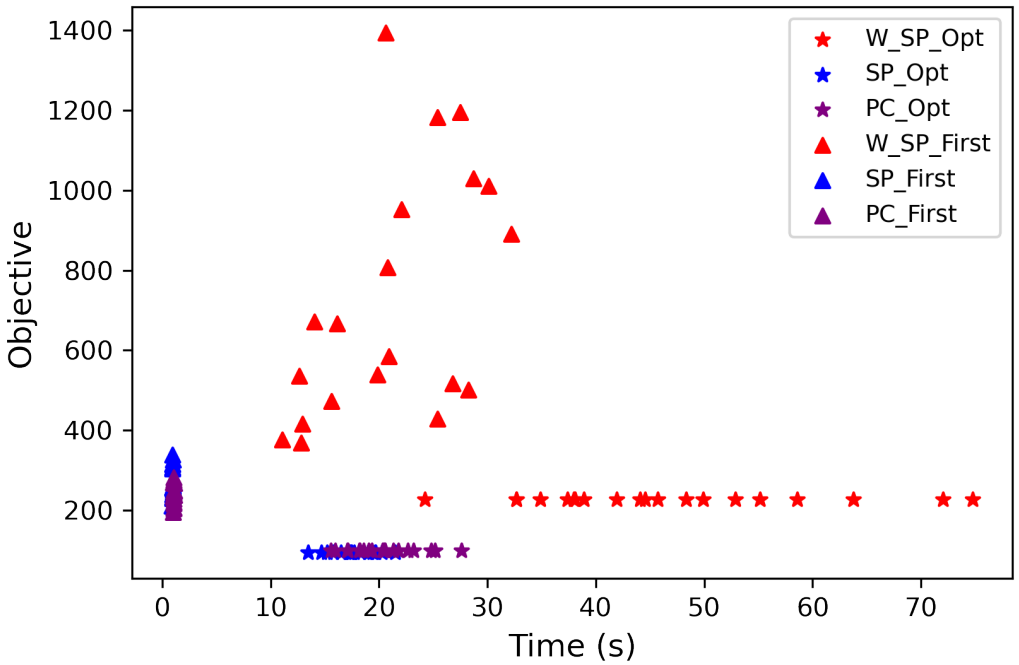


Figure B.51: Large layout, 15 vehicles, seed 1

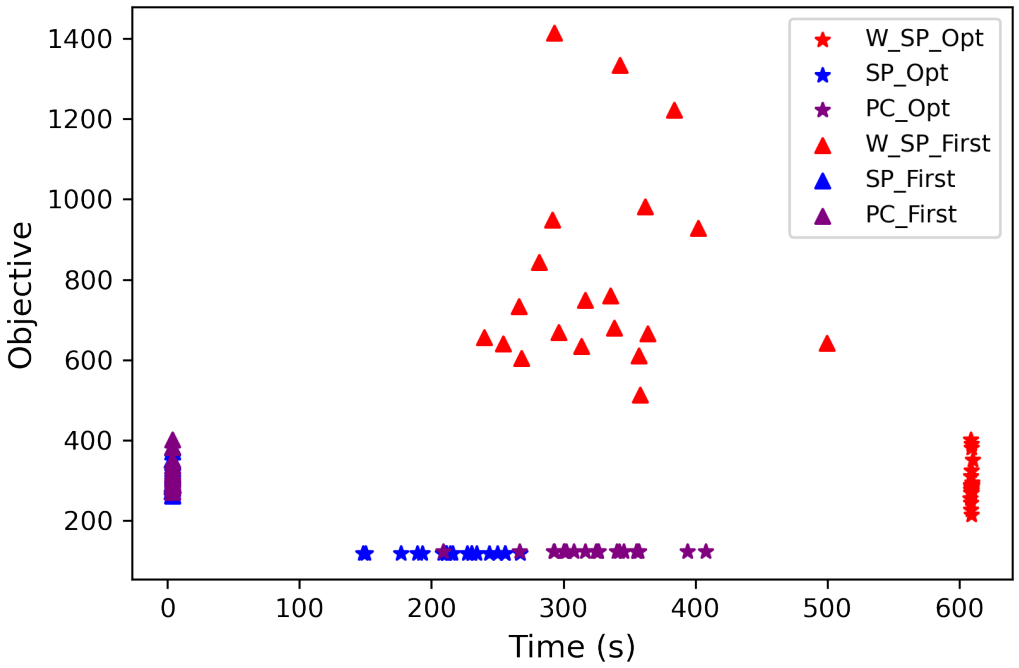


Figure B.52: Large layout, 15 vehicles, seed 2

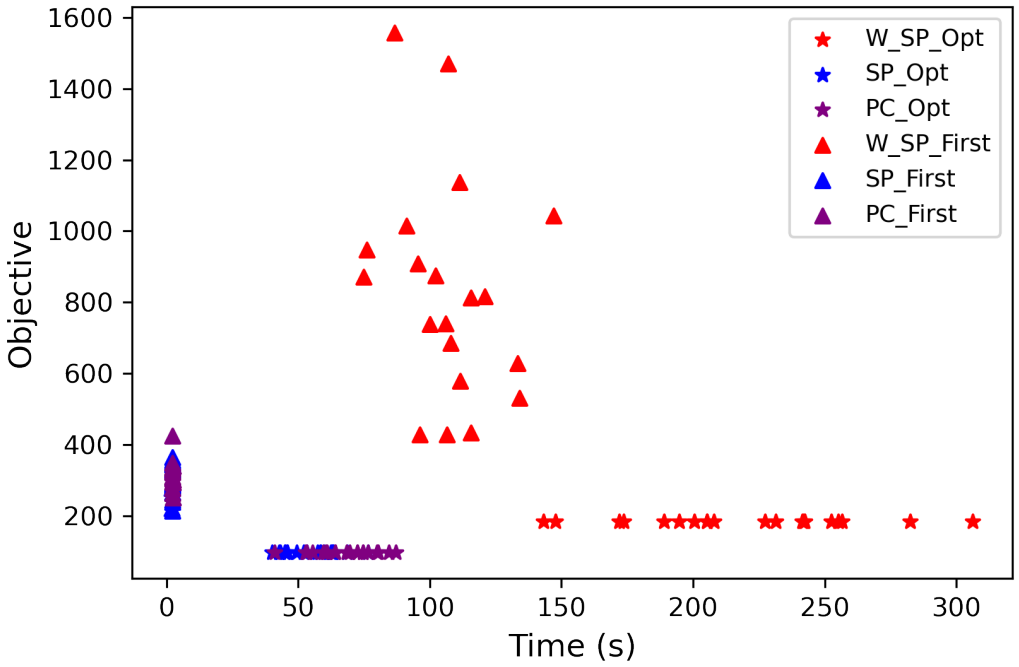


Figure B.53: Large layout, 15 vehicles, seed 3

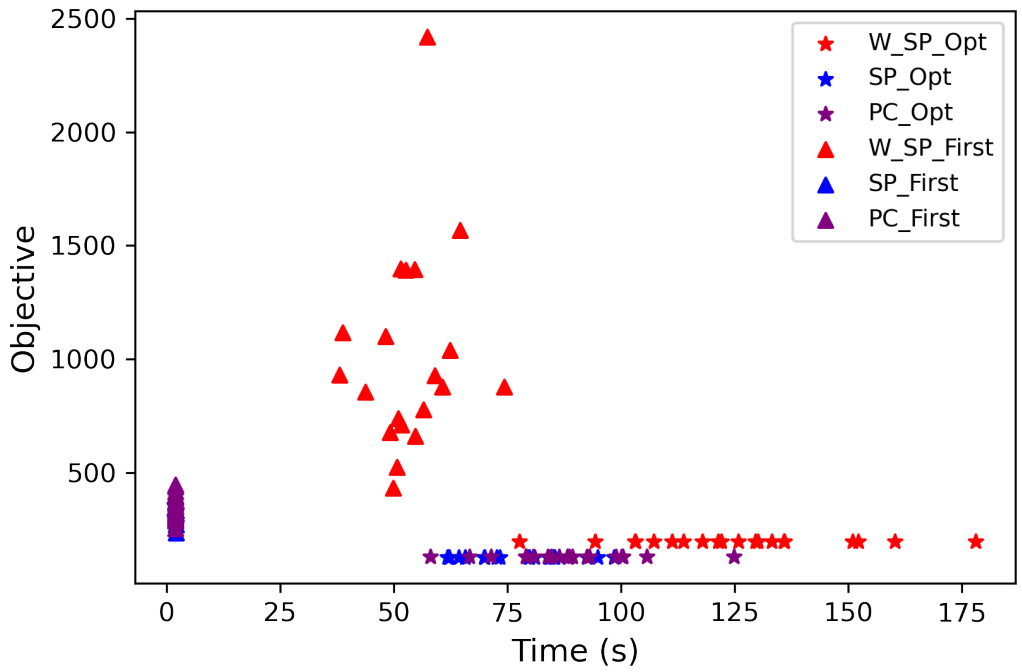


Figure B.54: Large layout, 15 vehicles, seed 4

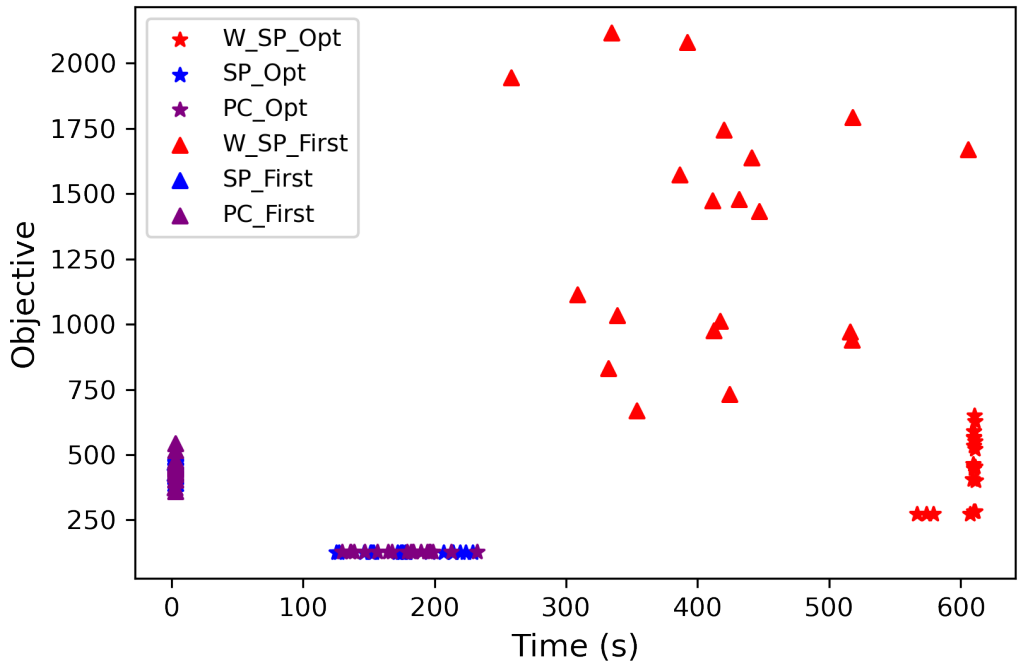


Figure B.55: Large layout, 15 vehicles, seed 5

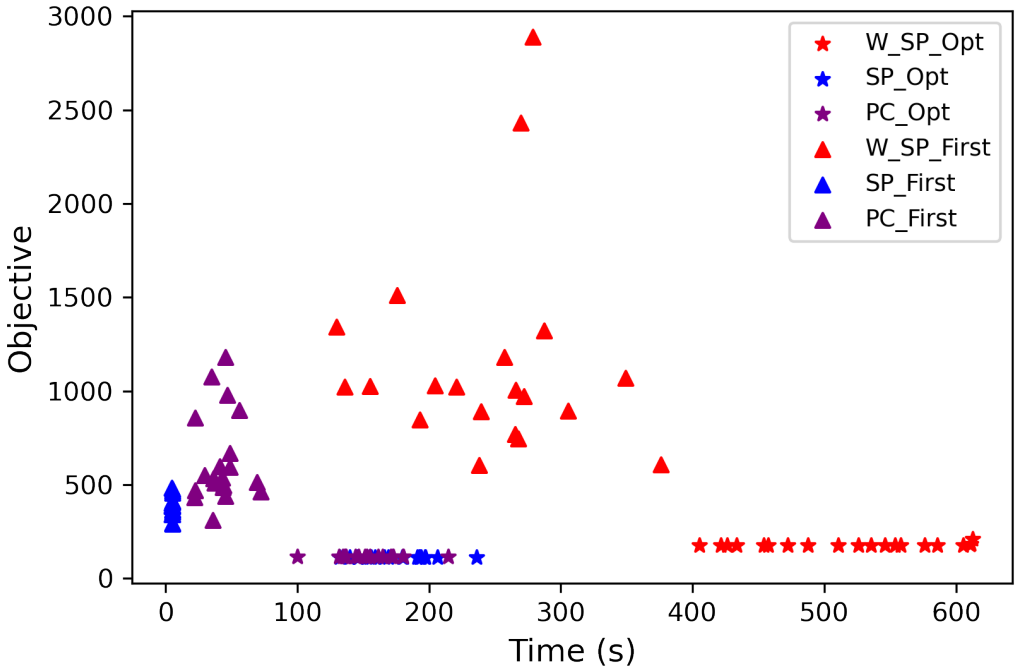


Figure B.56: Large layout, 20 vehicles, seed 1

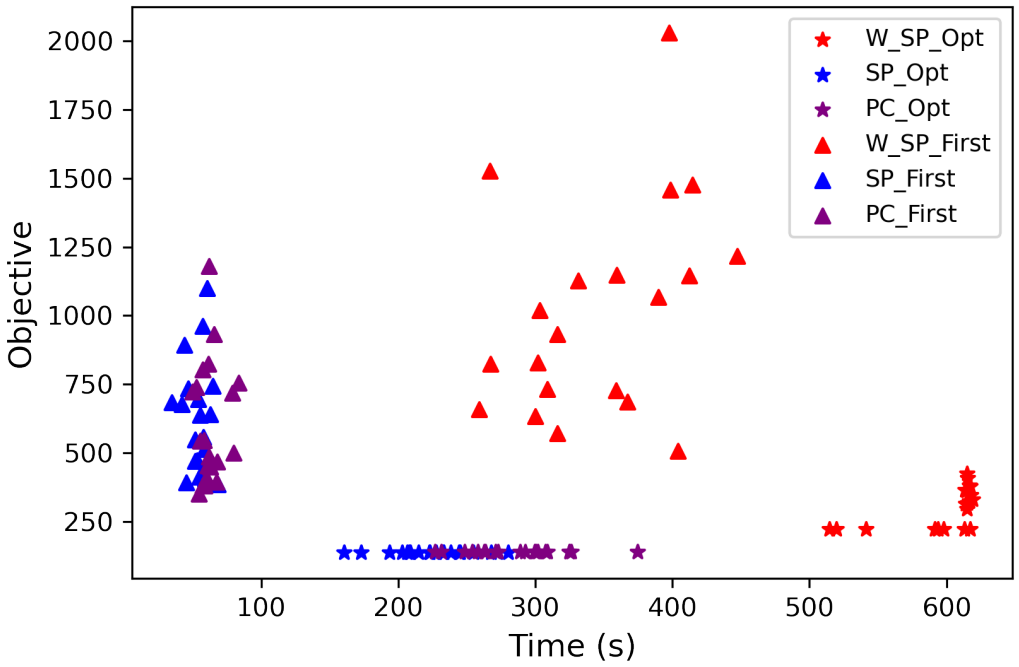


Figure B.57: Large layout, 20 vehicles, seed 2

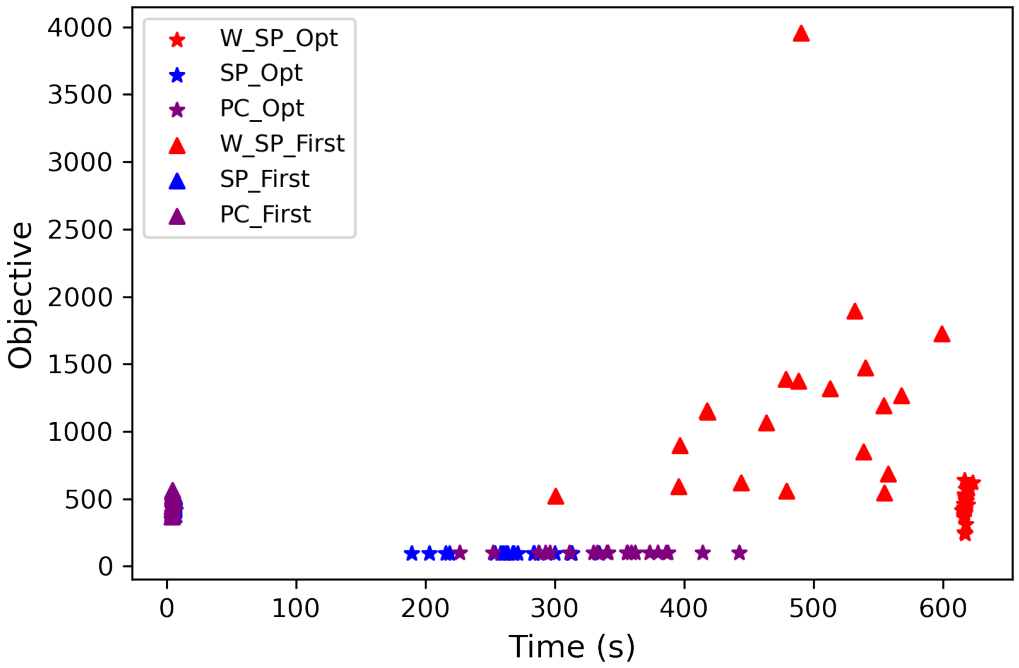


Figure B.58: Large layout, 20 vehicles, seed 3

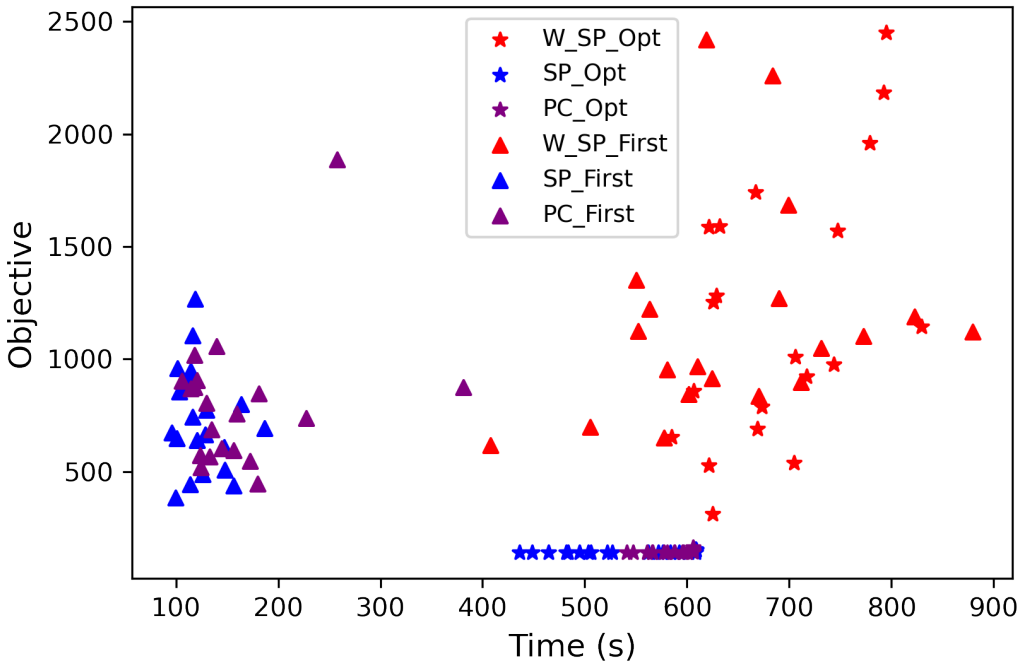


Figure B.59: Large layout, 20 vehicles, seed 4

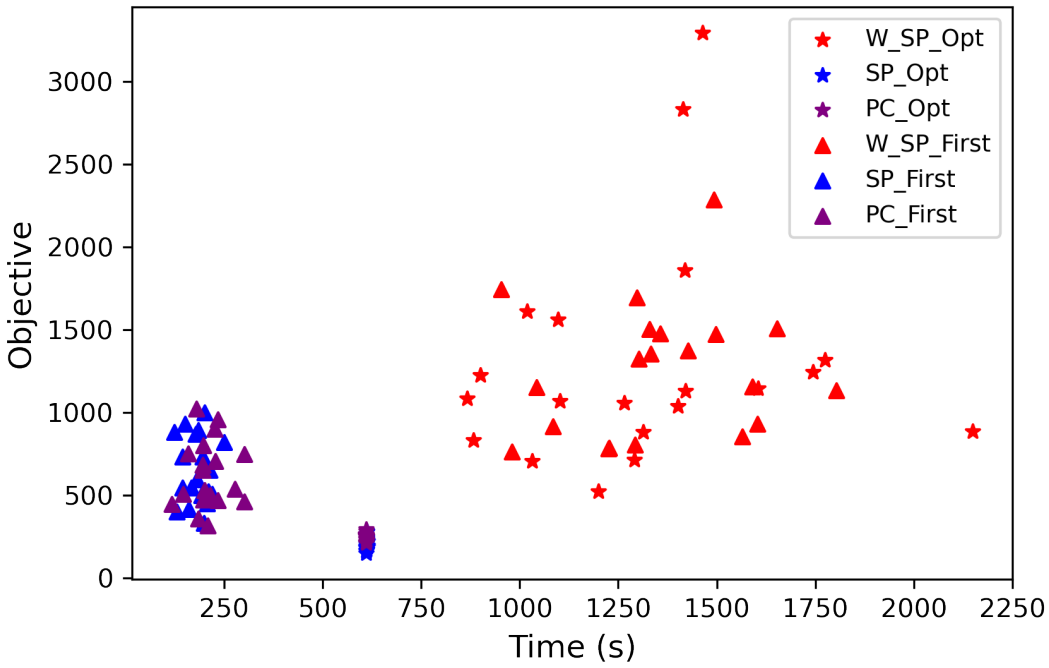


Figure B.60: Large layout, 20 vehicles, seed 5

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY