Designing and Planning in the Concurrent Action Realm for Resource Iterations

Dvir Cohen
dvir.cohen@campus.technion.ac.il

Shir Turgeman
shirturgeman@campus.technion.ac.il

November 24, 2024

Abstract

The Concurrent Action Realm for Resource Iterations (CARRI) is a planning framework aimed at addressing multi-agent coordination, resource management, and adaptability in dynamic environments by combining classical planning techniques with a genetic algorithm. CARRI's modular design offers flexibility across a range of semi-structured problem domains, making it suitable for managing concurrent actions and iterative updates without being overly domain-specific. This paper presents the architecture of CARRI and provides an evaluation of its genetic planning capabilities compared to other search-based approaches.[1]

1 Introduction

Modern logistics and operations involve complex planning problems that require coordinating multiple agents, managing resources, and adapting to changing environments. Consider a company managing daily package deliveries across different settings. In large warehouses, autonomous robots move along rails to sort packages efficiently. Outdoors, mini trucks or drones deliver packages while navigating urban areas. In fulfillment centers, forklifts and smaller robots work together to manage inventory and dispatch products. These scenarios demonstrate the challenge of managing concurrent actions, real-time updates, and resource allocation under varying conditions.

Traditional planning methods, like those using the Planning Domain Definition Language (PDDL), often fall short when dealing with dynamic environments requiring frequent updates and multi-agent collaboration. To address these limitations, this paper introduces the Concurrent Action Realm for Resource Iterations (CARRI). CARRI is a planning framework designed to handle complex multi-agent interactions

[1] Access the project Git repository at the following link: CARRI GitHub.

and adapt to evolving resource requirements, making it suitable for scenarios like urban delivery networks and warehouse management.
The framework integrates genetic algorithms to iteratively refine solutions, balancing exploration and exploitation. This approach allows the planner to adapt to changing environments while optimizing resource usage. CARRI combines elements of classical planning with flexible solution methods, making it applicable to a range of problem domains in logistics and beyond.

2 Related work

Over the years, researchers have developed a wide array of approaches to address the challenges described previously. The development of Vehicle Routing Problems (VRPs) and their numerous variants has been particularly influential in advancing methods for solving intricate logistical challenges, providing valuable inspiration for the Concurrent Action Realm for Resource Iterations (CARRI). The VRP family encompasses diverse real-world scenarios, from traditional vehicle routing to more specialized cases, such as dynamic routing with time windows, multi-agent systems integrating drones and trucks, and urban logistics constrained by environmental regulations. These problems underscore the necessity for adaptive, scalable, and efficient planning mechanisms, closely aligning with the goals of the CARRI framework. This section reviews contributions in this space and situates the CARRI domain within the broader research landscape.

2.1 Classical and Extended Planning

Classical planning methods, such as those based on the Planning Domain Definition Language (PDDL), have long been the cornerstone of AI planning research (Ghallab et al., 1998). These approaches are particularly effective in static, deterministic environments where actions have well-defined preconditions and effects. PDDL allows for systematic modeling of tasks, enabling planners to compute solutions by utilizing search algorithms based on an initial state and a desired goal state.
Classical methods face significant challenges in dynamic and uncertain environments. To address these limitations, researchers have extended PDDL in various ways:

- Temporal PDDL (PDDL2.1): Introduced by Fox and Long (2003), this extension allows modeling of actions with durations, enabling planners to handle scheduling tasks and overlapping activities.
- Numeric Constraints: Younes and Littman (2004) added numeric fluents, enabling planners to consider continuous resource management, such as fuel consumption or inventory.
- Multi-agent Extensions: Brafman and Domshlak (2008) developed frameworks to model the coordination of multiple agents acting concurrently, focusing on coordination and managing conflicts between agents sharing an environment.

However, while these extensions have broadened PDDL's applicability, real-time adaptability remains challenging, particularly in scenarios with frequent state updates.

2.2 Genetic Algorithms in VRPs

Recognizing the limitations of classical methods, researchers have explored hybrid approaches that combine structured planning with heuristic-guided optimization. Genetic algorithms (GAs), inspired by evolutionary processes, employ mechanisms such as fitness evaluation, selection, crossover, and mutation to iteratively improve potential solutions. GAs are particularly well-suited for planning problems characterized by large, dynamic solution spaces, making them especially effective for vehicle routing problems (VRPs) and related domains.

- Prins (2004) used GAs to optimize VRPs by encoding solutions as sequences of routes and evolving them through crossover and mutation, balancing exploration and exploitation to improve efficiency.
- Pillac et al. (2013) explored the use of GAs in dynamic VRPs, showing their effectiveness in adjusting routes in response to real-time inputs, such as traffic updates or new delivery tasks.

In these studies, GA populations consist of chromosomes representing delivery routes. In contrast, the CARRI framework encodes chromosomes as sequences of actions over a planning horizon, evaluating them based on the overall feasibility and effectiveness of the resulting plan. This approach provides flexibility across domains while retaining the adaptability inherent in GAs. Details of this approach are provided in later sections.

3 The CARRI Framework & Modeling

3.1 Vision

The CARRI framework strives to allow designing many different versions of VRP problems, each with their own niches, rules, constraints and twists. Those problems can be applied by different models which may have their own rules for how turns and iterations are applied and by which constraints. The versatility of CARRI framework allows for the research of different planners in a manner that is dependent on the realm rather than on the domain, simulating a real-life model, or instructions for real vehicles based on reduction of a problem.

3.2 Positioning CARRI vs other planning design methods

We decided to create the CARRI framework as we have found the commonly used tools (such as PDDL), while powerful – unfit for our purpose. While PDDL with extensions supports concurrent planning, it doesn't support situated settings with items being able to be added or removed, requiring work arounds accommodated for the features which are core in our framework. Moreover, our framework has been built with VRP-like settings in mind, to allow for different planners through different models to be able to solve different models, unlike the PDDL format which is more general purpose and lacks the specifics of the realm. However, there are few places where our model lacks compared to PDDL. Our model lacks the tools to represent

many complex conditions and constraints which PDDL may have, as well as our restraint in the realm of discrete deterministic problems. Furthermore, while we managed to represent some complex scenarios, a furthermore complex domain may demand the usage of a fully unrestricted programing language instead.

3.3 Concept & domains characterization

Our framework is designed for creating VRP-like problems in a discrete situated setting with the following characteristics:

1. The world is discrete, deterministic, graph based, with partial observability for the long term. The goal is to reduce costs over time.
2. The world consists of entities, mainly of location, vehicle, package and request types.
3. At each turn, all vehicles are conducting actions concurrently, traveling usually through locations on a graph (that can be weighed), picking up packages and delivering them to requests of the same type. Each vehicle will conduct exactly a single action per turn, which may be of 'travel', 'wait', 'pick', 'deliver', or any other type of action.
4. Actions are usually applied sequentially, where actions of vehicles may conflict with the actions of the following vehicles (such as two vehicles trying to pick the same package).
5. Environmental steps may be applied after the vehicles' action at the same turn.
6. Vehicles' action and environmental steps may have a cost attached to them.
7. There will be an explicit cost related to the existence of packages or requests, attached to an environmental step, resembling the need to deliver them.
8. After a certain number of turns, a new iteration will begin. A domain may have an iteration step which will change the state of the world, and new entities may be added to the world.
9. The planner may not know how many states are upcoming, and which entities are to be added (if at all) by the start of new iteration.

3.4 Domain design

A problem is usually designed by CARRI script format on a .CARRI file (inspired by the way PDDL is formatted and written). In the Domain section (3.7), the entities, variables, actions, environmental steps and iteration step will be defined. In the Problem section (3.8), the number of some of the entities and the initial values will be defined, as well as entities to be added to the world by upcoming iterations.
Some worthy notes to take are that CARRI separates between types of entities which can be added / removed, such as packages and requests, and types of entities that aren't, such as locations (In the case of vehicles and other types of entities, this is up

to the domain's design). Entity types that are addable / removable are sometimes referred to as 'items'.
There is also a distinction between variables which are constant (such as adjacency or type of location), varying (such as location of vehicle), or related to items (such as location and type of packages). In the case of items related variables, there may be defined nuance whether the item is changeable or not within the scope of an iteration. Variables are usually of integer data type. Entity types and entity identifiers are internally referenced by non-negative integer value (starting at 0), and usually other variables may be of integer data type. Such examples may be 'capacity', 'fuel', 'weight', present thought different domain. However, the CARRI format also supports boolean data type, and map and set data types (referred as MATCH and MULTY) for constant non items variables (However, those variables may only contain integers or booleans).
Entities, variables and actions may have a 'base' type:

- Entities belong to a predefined categories of 'Location', 'Vehicle', 'Package', and 'Request'.
- Variables classified into 'adjacency' (associated with Location entities), 'location' (indicating an entity's position), 'entity' (denoting the type of position), and 'type' (related to the package or request type for matching purposes).
- Actions include the fundamental operations of 'travel', 'wait', 'pick', and 'deliver'.

While defining base types is not always mandatory, only entities of the 'Vehicle' type are expected to perform actions. Some planners may rely on base types, and as such, it is worthwhile to indicate base types.
Actions are defined by 6 key sections:

- Parameters: each parameter is specified by name and by its entity type. The first parameter is always the acting vehicle's parameter.
- Base Action – Optional, indication of base action
- Preconditions – list of conditions to validate upon a state to determine if the action can be applied with given parameters.
- Conflicting Preconditions – additional list of preconditions, also mandatory to validate upon a state. They differ from 'normal' preconditions as they are expected to be affected by previously applied actions of other vehicles, giving sufficient indication that action can't be applied as of previous vehicle's actions.
- Effects – the effects of the action upon the state, with the possibility of affecting different entities.
- Cost – cost of the action.

Not all actions may implement some or all of the 4 latter sections - For example, wait actions usually don't have any of them. The implementation of Conflicting Preconditions and Cost usually varies. If an action has no Cost section, its cost is effectively zero. Another feature that CARRI provides is the ability to 'inherit' action, with certain limitations.

3.5 Parsing

Using a specified parser, CARRI Domain and Problem files are parsed into Simulator, Problem and State objects, as well as an iteration list.
The simulator is the central to managing state transitions and coordinating actions, the simulator integrates static data from the Problem class and dynamic data from the State class. The simulator may be used to generate valid actions, successor state, apply transition of actions upon a state, create a string representation of an action, and so on.
The problem encapsulates static data, such as entities, variables, and indexing information, serving as the backbone for state interpretation and simulator queries.
The state focuses on dynamic data management, as it encapsulates it. This enables transitions without requiring constant access to static problem definitions.
The iteration list is essentially a list of items (entities) to be added by each upcoming iteration (items at first list are added by the beginning of iteration '1' – the second iteration). Each list may include all or some of the items types or no items at all. While a domain aware model may add entities by its own, either deterministically or randomly, the iteration list allows for a way to deterministically add entities without domain knowledge, to consistently run across multiple tryouts.

3.6 Model

We implemented a model that utilizes the CARRI framework. Our model consists of Business, Manager and Planner with the possible addition of Search Engine and Heuristic (utilized by Planner).
Our model simulates 'delayed planning'. Given n – number of transitions (turns) per iteration, and t – iteration time, we assume it takes n/t seconds to apply every discrete action for every vehicle. By each iteration, the manager calls the planner to provide a plan of n transitions by t seconds to provide for the upcoming t seconds. By the end of the planning time, the manager provides the given plan for the business and retrieves from the business the state of the world, which is expected after t seconds, with the addition of newly available items.

For example, with $t = 120$ and $n = 10$. The current time is 12:00 pm. The manager provides the planner with the expected state of the world by 12:02 (120 = 2 minutes) and expects the planner to compute a plan consisting of 10 transitions (10 actions per available vehicle). By 12:02 the manager retrieves the requested plan from the planner. It provides the plan to the business object, which we assume is given to acting vehicles. The business also calculates the expected state of the world by 12:04, after applying the plan, the iteration step, and adding items – entities which will be available by that time (or are available now, but the plan doesn't accommodate now). The manager retrieves the calculated expected state of the world. Assuming insignificant time has passed (we are still at 12:02), the manager provides that calculated state back to the planner, expecting it to compute the next plan that will be executed by 12:04, and so on.

We may describe the roles of each object as follows:

1. **Business:** Executes the plan provided by the manager, applies transitions, and updates the internal state while handling dynamically added items and iteration steps

2. **Manager:** Coordinates between the planner and business, oversees plan execution, manages timing, and ensures external updates are handled.

3. **Planner:** Generates plans by using available state information. May use search engines and heuristics to evaluate states and optimize the plan.

4. Additional objects:

    - **Search Engine:** If used by the planner, the search engine can take over the role of generating and optimizing the plan. The planner initiates the search engine by setting up initial conditions, and the search engine then refines and produces a viable plan.,
    - **Heuristic:** Used by the search engine or planner to evaluate the quality of states and guide the planning process towards efficient solutions.
    - **Assigner:** A specialized tool, used to quickly generate one or more valid potential plans within given constraints and instructions. Currently this role is not generalized and implemented only by Partial Assigner.

While the model suggests that it runs indefinitely, or at least for a long period of time, we only test it for a limited number of iterations per problem, assuming the accumulative score indicates performance for the long term.

3.7 Domains

Currently we have implemented 4 unique domains:

1. **Trucks and Drones:**
   This domain consists of 2 vehicle type – Drones and Trucks. They operate in an urban city, represented as a graph. The locations can be one of 3 types – 0: roads, travelable by every vehicle, 1: corridors, travelable by drones only, 2: bridges (or tunnels), travelable by trucks, inaccessible to drones without boarding a truck.
   The drones are expected to deliver packages to requests of fitting type, both can be located on a road or in a corridor. The drones have a limited charge, sufficient for only 10 turns without a charge. The drones can charge by boarding a truck, or by emergency charge after depleting it completely (which is much costlier). The drones can board a truck to travel through the city, including across bridges. The Drones may or may not leave packages on the trucks. Drones can carry only one package at a time (when not boarding a truck). A truck may carry up to 4 drones and up to 8 packages at a time (capacity of each type isn't dependent on the other).
   The truck can travel across one location at a time, or turbo travel across two locations (assuming both aren't a corridor) at additional cost. By the end of each iteration, all existing requests' impatience is increasing (from 0 to the limit of 3), making it so not delivering them is costlier.

2. **Cars:**
   Cars travel through the city, expected to carry up 2 packages and deliver them to requests. The city is represented by a weighted graph, while cars can reach locations of certain distances from their own location, it takes larger amount of gas and cost to reach farther locations. The cars have fuel, up to 40. Whenever their fuel level is not full, they can fuel in one turn up to maximum. The cost of fueling is the number of fuel shortage. If the location where they decide to fuel is not of station type, the cost of fueling will be 4 times costlier. The cost of each package not delivered is consistent per each turn.

3. **MotorCycles and Letters:**
   Mailing company deploys self-employed motorcyclists with no capacity and different availability across the neighborhood to deliver letters of certain types to moving people. By the end of each iteration, each motorcyclist's availability depletes by one, and if it reaches 0, the motorcyclists end their shift, dropping every letter they have on hand at the location. The company receives additional motorcyclists every now and then, but they make sure they always have at least 3 available before the addition of more motorcyclists. The letters have costs, depending on letter type (2 + 2 * letter type). The people who request the letters are not static, they move across predefined 'high route' and 'low route', changing their route choice every few turns (deterministically).

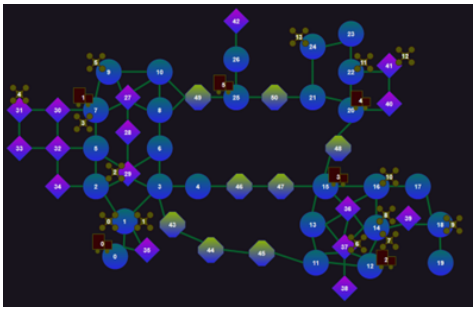
4. **Rail System Factory:**
   With some inspiration by one of the authors' temporal work at aluminum profile factory by the automatic warehouse. In this factory, carriages are traveling across a rail system, requested to deliver weights of profiles to orders. The carriages always travel across exactly 2 rails to travel from one station to another. For safety reasons, only 1 carriage may use a rail and only up to 3 carriages may accommodate a station at any given turn. Each package of profiles is measured by kg, and an order may be fulfilled up to its requested weight of requested type. the carriages can carry up to 50 kg of profiles. If two packages with the same type reside on the same carriage, the carriage can take a turn to combine them. Certain stations are not receiving packages from the outside and the carriages can store on them up to 99 kg of profiles. The cost of each turn is the total weight of requests not fulfilled.

All our problems take place in undirected graph worlds, but it is very possible to design a domain with directed graph world. However, it might not be currently possible to design a problem with negative weights.

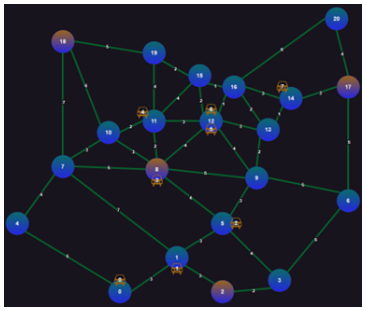
3.8 Problems

We created 7 sample problems for the 4 implemented domains.
Below is a short description of the problems, with our assessment of problem difficulty considering number of entities, iterations and observed costs.

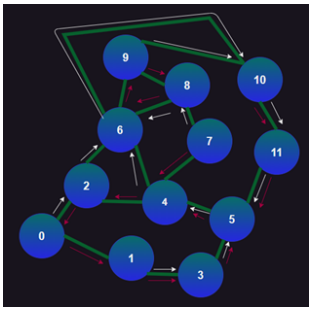
- **Trucks and Drones:**



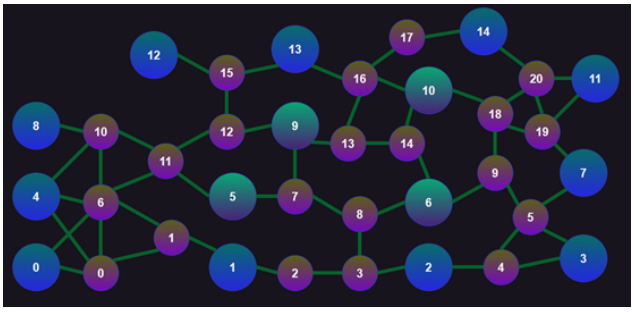
(a) Trucks and Drones 3: Blue circles are roads, purple squares are corridors, yellow octagons are bridges.



(b) Cars 2: Orange locations are (fuel) stations, distances (weights) are represented by the numerical value of the edges.



(c) MotorCycles and Letters 1: White arrows represent the 'high route', and the pink arrows represent the 'low route'.



(d) Rail System Factory 1: Large circles are stations, and small circles represent rails. Blue for regular, green for storage.

Figure 1: All figures illustrating different domains in CARRI.

1. **Trucks and Drones 1 -** first implemented problem, with 5 iterations, 8 locations, 3 drones and 2 trucks, is a relatively simple problem.
2. **Trucks and Drones 2 -** simple extension of first problem, with the only change is 8 drones and 4 trucks. The increase in the number of vehicles makes it slightly a harder problem, but it won't be considered as hard.
3. **Trucks and Drones 3 -** relatively a very hard problem, requiring a well-coordinated plan. It has 13 iterations, 51 locations, 14 drones and 6 trucks. The problem was inspired by New York, having 'islands' divided by bridges (so drones can cross sections only by boarding trucks). Figure 2a.

- **Cars:**

1. **Cars 1** – relatively simple problem, with 5 iterations, 3 cars and 10 locations.
2. **Cars 2 -** a relatively hard problem, with 18 iterations, 8 cars and 21 locations. Model of the map takes certain inspiration from a sitemap of London. Figure 2b.

- **MotorCycles and Letters:**

1. **MotorCycles and Letters 1** – relatively easy problem, with 6 iterations and 12 locations. While the number of motorcycles is changing It seems as if the number of motorcycles doesn't exceed 5. Figure 2c.

- **Rail System Factory:**

1. **Rail System Factory 1** – relatively easy to medium in difficulty, with 9 iterations, 15 stations (and 21 rails) and 6 carriages. Some may say the model of the map has some resemblance to a grid. Figure 2d.

3.9 Competition - different Planners

The competition between planners in this study centers on comparing a Genetic algorithm-based planner with two search engine-based planners, each utilizing a different search engine. All three planners inherit from a common base, the 'AssigningPlanner', which provides shared initial functionalities. Each planner, regardless of the method used for refining the plan, begins by generating an initial plan using the Partial Assigner. This ensures that all planners can provide a valid solution, even for challenging problems with a short runtime (such as 10 seconds). The commonality lies in their shared strategy of creating a baseline plan and then utilizing their specific methods to improve upon it, whether through genetic optimization, UCT-based search, or other approaches.
The search engine-based planner can utilize different search engines, including:

1. **UCT:**
   The UCT search engine used within this Model adapts classical Upper Confidence Bound for Trees (UCT) to tackle the multi-vehicle planning problem. Instead of generating all possible successor actions, which would lead to an overwhelming number of combinations, it generates a subset of actions using a partial assigner. Rollouts are conducted with a fixed number of transitions in iteration steps, unlike traditional UCT, which may have variable rollout lengths. This method is used to gradually explore more outside the compound of the iteration.

2. **Greedy:**
   The Greedy search engine, while simpler in concept, also proved to be an effective competitor. It starts by generating transitions for a fixed number of steps, with a focus on finding the plan that minimizes average cost. If better plans are found within a given time frame, the search expands to explore additional steps, gradually expanding exploration outside the iteration compound. This approach maintains a balance between efficient search and exploration, leading to surprisingly competitive performance compared to more complex methods.

3. **IDAStar:**
   An attempt at implementing the IDAStar algorithm into search engine. The IDAStar search engine employs a heuristic to guide the search process, using iterative deepening and cost-based bounds to explore potential plans. It starts

with a heuristic value to set the initial bound and then searches for solutions that fall within that bound, increasing it iteratively as needed. This search method proved to be disappointing with low performance, and thus was not tested. That may be due to lack of implemented generalized, strong heuristics, an aspect which might be developed more in the future.

3.10 Genetic Planner

We implemented a genetic planner within the CARRI framework that leverages the genetic algorithm logic to planning in our realm. Unlike traditional VRP-based GAs that focus on assigning customers to the most suitable vehicle while respecting constraints, our Genetic Planner's population evolves sequences of actions for multiple agents over a planning horizon. we decided to take this different approach due to the nature of our realm, to maintain its flexability,
The Genetic Planner initializes a population of potential plans, each referred to as a chromosome, representing a sequence of actions for vehicles over a fixed planning horizon. The algorithm iteratively refines these chromosomes using GA techniques such as selection, crossover, and mutation to optimize efficiency. The Planner is comprised of the following:

- **Population Initialization:** The initial population is generated using a heuristic-based partial assignment planner, ensuring feasible sequences of actions over the planning horizon that greedy prefer sequences of actions that complete the task at hand - meaning prefes 'Pick' and 'Deliver' actions. This approach helps balance the exploration of new solutions and the refinement of high-quality ones.

- **Fitness Evaluation:** Chromosomes are evaluated based on several criteria: total cost, number of completed "Pick" and "Deliver" actions, collaboration bonuses for package transfers, and penalties for redundant or infeasible actions. The objective is to minimize costs while maximizing overall task completion and collaboration.

- **Selection and Mutation:** Tournament selection is used to choose parents for reproduction. Here, we differ from classical mutation methods due to the fact our chromosome is a sequence of actions, the validity of actions needs to remain whole after alternating it. Therefore, the mutation function truncates a parent at a random point and then refills the remaining sequence to the desired length (horizon). The mutation is done to maintain diversity and improve adaptability.

- **Elitness:** Elite solutions are preserved across generations to maintain high-quality chromosomes, ensuring that the best-performing individuals are retained.

- **Population Restarts:** If the best plan hasn't changed in 5 iterations, then the population completely restarts. This mechanism enriches diversity because the population size is very small compared to all possible sequences, Hence re-creation could help exploring.

4 Experiments

4.1 Metrics

As our main goal is to reduce costs over time, our main metric was the mean accumulative cost. However, we also measured the median and std of the accumulative cost. Besides that, we know that greater numbers of 'pick' and 'deliver' actions usually indicate better performance, as the domains in the realm are focused on vehicles delivering packages to requests. As such, we also measured the mean, median and std of the accumulated pick and deliver actions counts.

4.2 Method

The experiment was conducted through pipelines, updating a log file of .csv format. The experiments were ran on a testing computer within a similar environment (details on computer specification and environment in additional notes (section 6.2).
Each specified problem has been ran 5 or 6 times by every one of the following planners:

1. Genetic Planner (no search engine, no heuristic).
2. Search Engine Based Planner with UCT search engine (no heuristic).
3. Search Engine Based Planner with Greedy search engine (no heuristic).

Each experiment was conducted with the settings of 10 transitions (turns) and 120 seconds per iteration.

4.3 Results

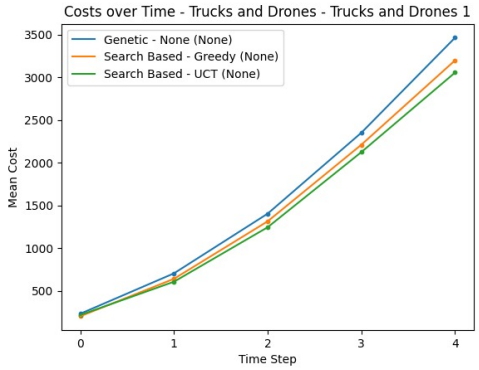
The following results are displayed in Table 1, Table 2 and Figure 2.More ummarization of Results of the rest of the problem can be found in the repository, inside the Logs file. As we can see from tables 1 & 2, Genetic Planner outperforms Greedy and UCT on the "hard" problems, in all measures (including CI).

Planner	Total Mean Cost	Total Mean Picks	Total Mean Delivers
Genetic	**42,464.666** (± 2,983.731)	**27** (± 2.097)	**14.333** (± 1.366)
Greedy	44,610.166 (± 3,233.687)	18 (± 3.741)	10.833 (± 2.714)
UCT	44,928.5 (± 3,309.522)	21.5 (± 2.167)	10.833 (± 2.041)

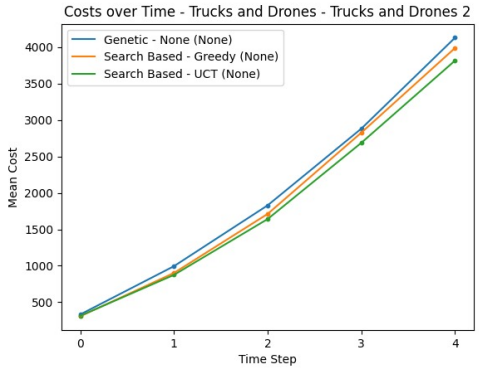
Table 1: Truck and Drones 3

Planner	Total Mean Cost	Total Mean Picks	Total Mean Delivers
Genetic	**21,553.166** (± 307.559)	**40.666** (± 0.516)	**38.833** (± 0.408)
Greedy	23,490 (± 1,588.136)	38.5 (± 1.048)	31.333 (± 2.422)
UCT	24,852.5 (± 1,333.361)	37.5 (± 1.516)	32.5 (± 1.870)

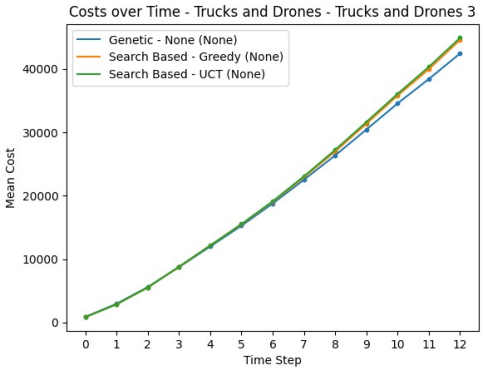
Table 2: Cars 2



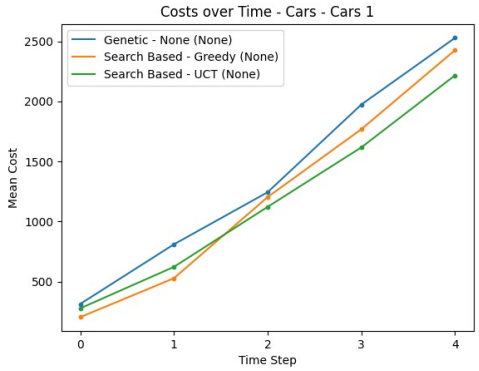
(a) Trucks and Drones 1



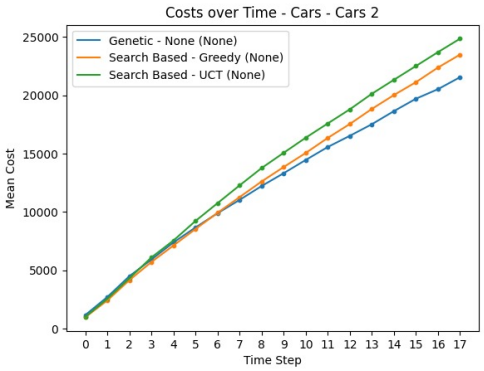
(b) Trucks and Drones 2



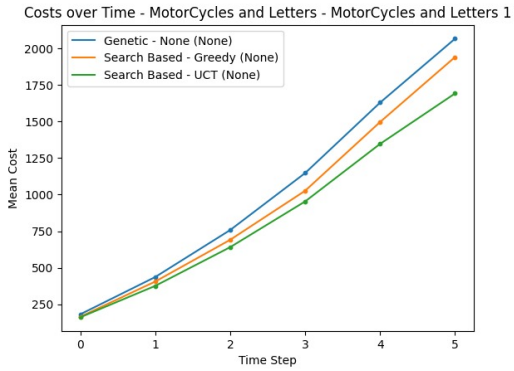
(c) Trucks and Drones 3



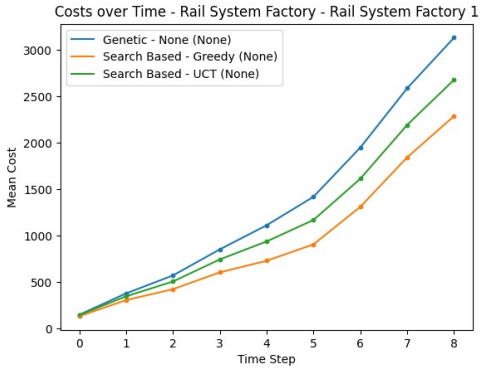
(d) Cars 1



(e) Cars 2



(f) MotorCycles and Letters 1



(g) Rail System Factory 1

Figure 2: Experiments Results

5 Conclusion and Future Work

The CARRI framework successfully integrates classical planning techniques with genetic algorithms to address multi-agent coordination and dynamic resource allocation problems. Through a combination of modular design and adaptability, CARRI demonstrates flexibility across various semi-structured domains. The experimental results indicate that the Genetic Planner, in hard problems and some others, outperforms traditional Greedy and UCT-based approaches in complex, high-concurrency scenarios, particularly in reducing total cost and improving task completion metrics.

Future work can focus on expanding the versatility of the CARRI framework to handle more complex domains and dynamic conditions, as well as adapting the framework to real vehicles for practical deployment. This includes improving the representation of continuous environments, extending the framework to accommodate probabilistic elements, and developing more sophisticated heuristics for enhanced efficiency. Additional efforts can also be directed towards improving scalability and exploring the integration of other evolutionary and hybrid methods to further enhance the planning capabilities of the CARRI framework.

6 Additional Notes

6.1 Repository

Access the project Git repository at the following link: CARRI GitHub.

6.2 Computer Specifications & Experiment Environment

The computer in use of the experiment was a Gigabyte g5 kc model with the following specifications:

- Windows 11 Pro operating system,
- Processor: Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz 2.50 GHz
- Ram: 16.0 GB (15.8 GB usable)
- System type 64-bit operating system, x64-based processor
- Graphics card: NVIDIA® GeForce RTX™ 3060 Laptop GPU 6GB GDDR6

The experiments were ran using Python 3.12 version.
The experiments were conducted with the computer charged, no additional inputs attached, no non essential programs running in parallel, usually after shutdown / restart of the computer.
The experiments were conducted with the computer on 'flight mode' (no internet or Bluetooth services activated).

7 References

1. Ghallab, M., Nau, D., & Traverso, P. (1998). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

2. Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20, 61–124.

3. Younes, H. L., & Littman, M. L. (2004). PPDDL1.0: An extension to PDDL for probabilistic planning. *Proceedings of ICAPS*.

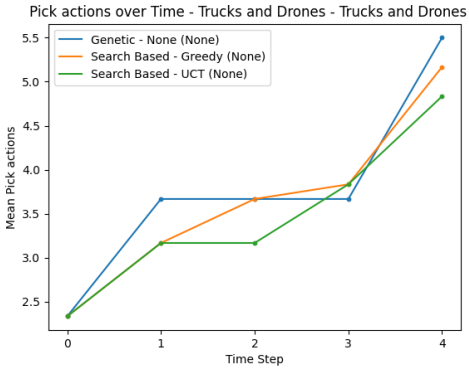
4. Brafman, R. I., & Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. *Proceedings of ICAPS*.

5. Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12), 1985–2002.

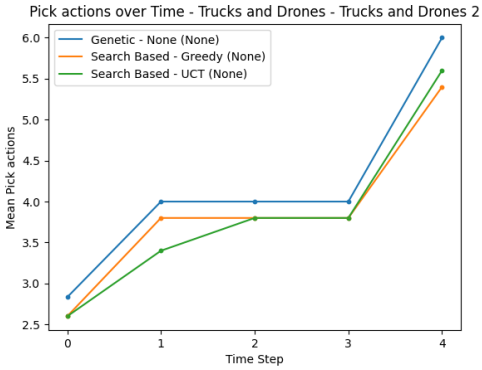
6. Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1), 1–11.

8 Appendices

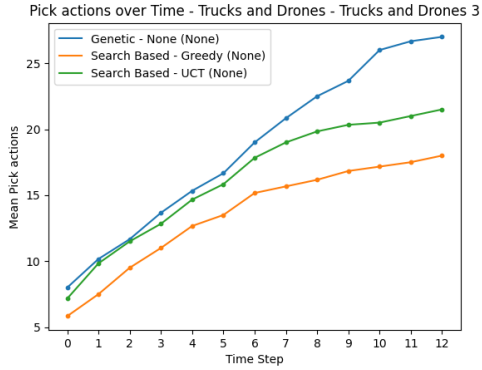
Below we present more results for all 7 problems.



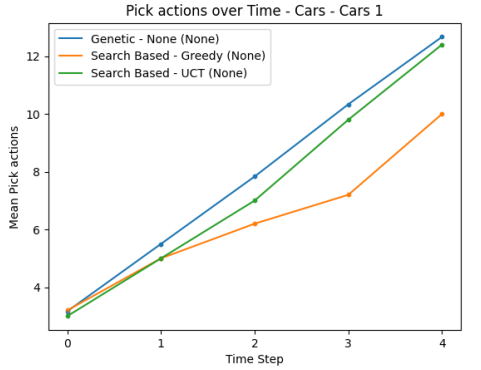
(a) Trucks and Drones 1



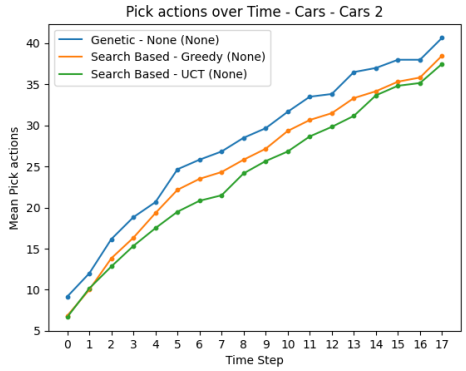
(b) Trucks and Drones 2



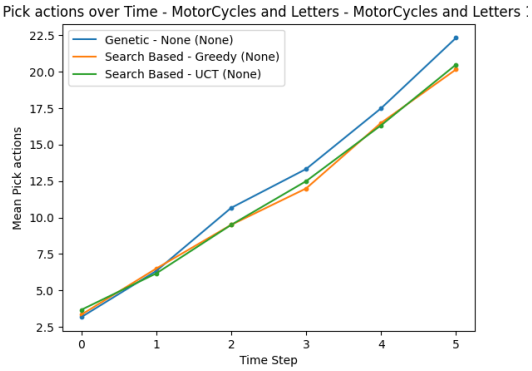
(c) Trucks and Drones 3



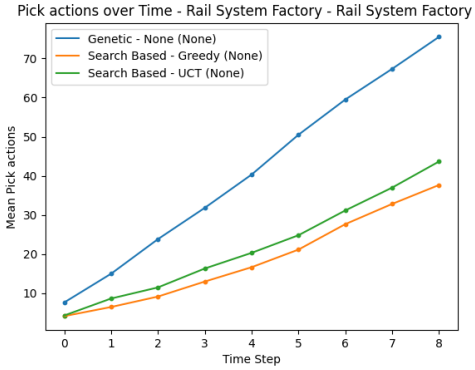
(d) Cars 1



(e) Cars 2

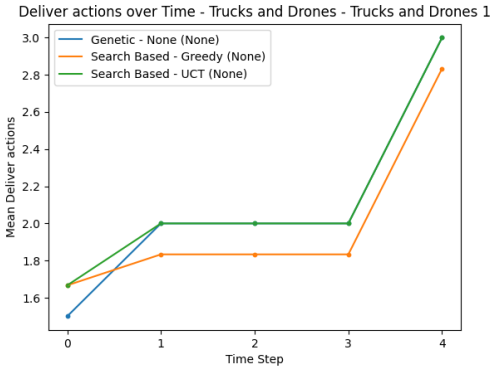


(f) MotorCycles and Letters 1

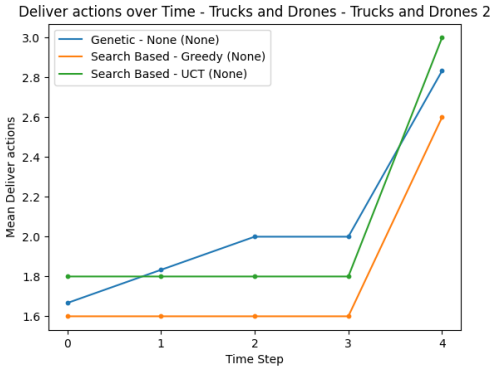


(g) Rail System Factory 1

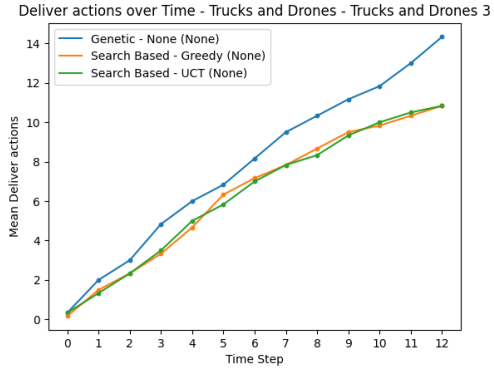
Figure 3: Results for Total Mean Picks across all problems.



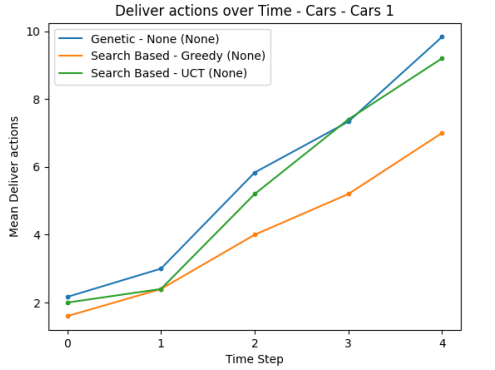
(a) Trucks and Drones 1



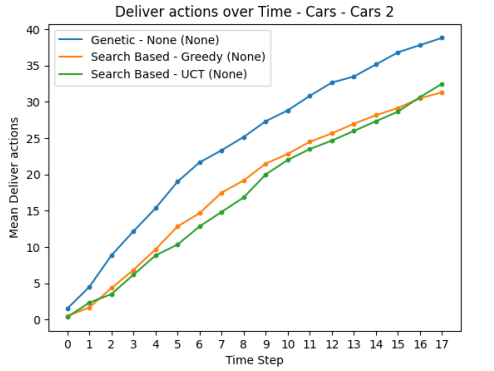
(b) Trucks and Drones 2



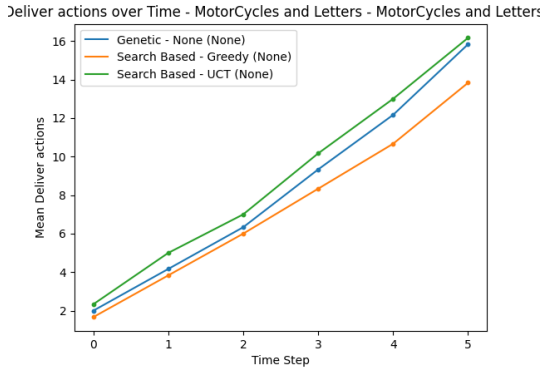
(c) Trucks and Drones 3



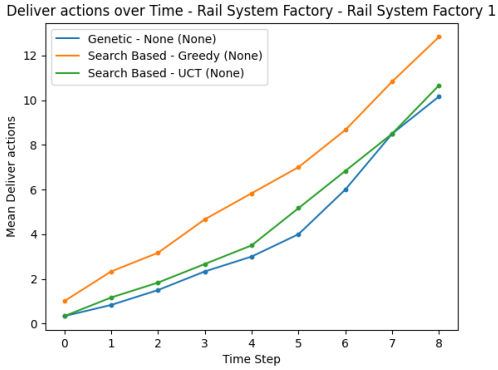
(d) Cars 1



(e) Cars 2



(f) MotorCycles and Letters 1



(g) Rail System Factory 1

Figure 4: Results for Total Mean Delivers across all problems.