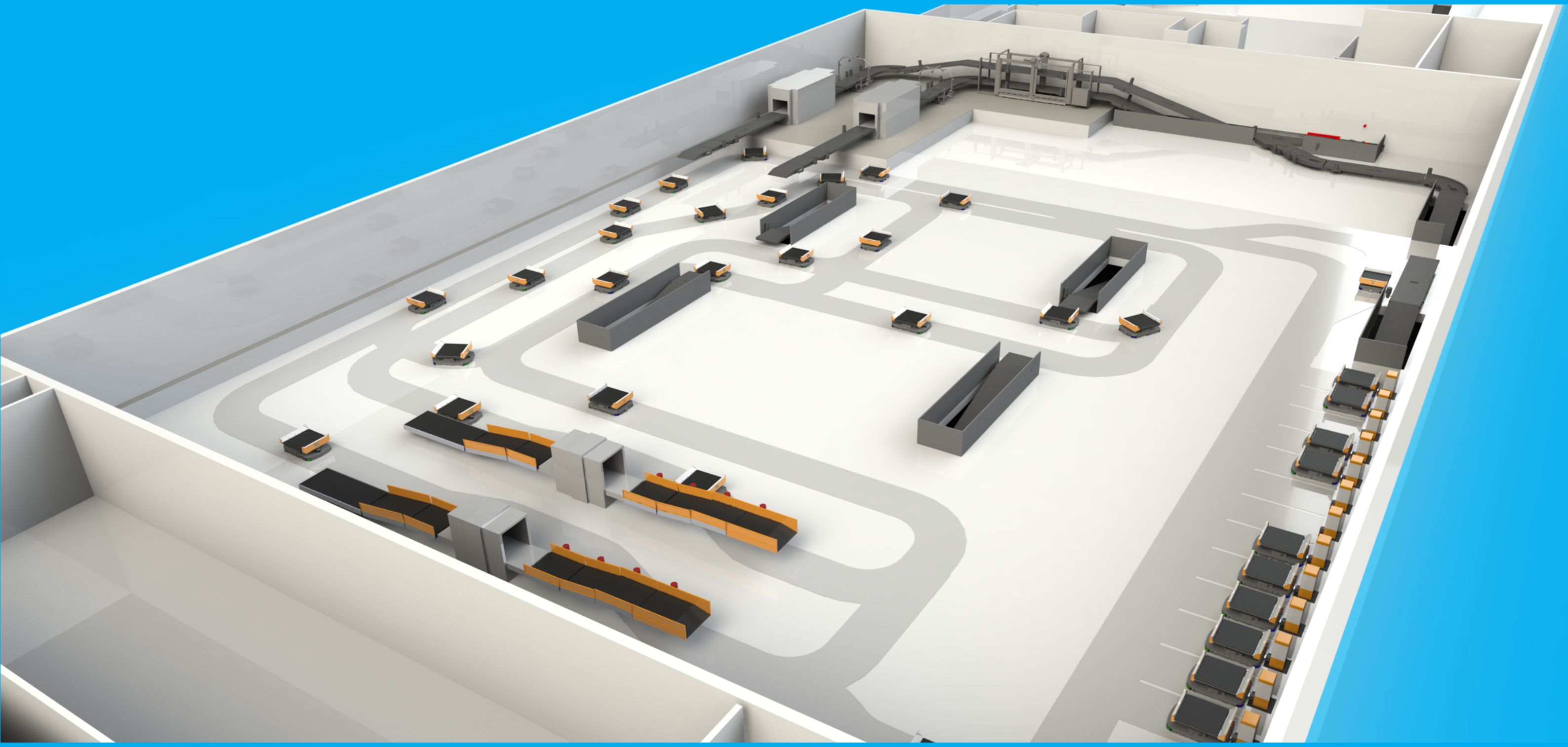
Automated Guide-Path Map Generation for Automated Guided Vehicle Systems

L.L. Endlich

An Algorithmic Framework



TU Delft

Automated Guide-Path Map Generation for Automated Guided Vehicle Systems

by

L.L. Endlich

to obtain the degree of Master of Science in Robotics
at the Delft University of Technology

Student number:	5428246	
Thesis company:	Vanderlande	
Project duration:	March 1, 2022 – December 19, 2022	
Thesis committee:	Dr. ir. J.C.F. de Winter,	TU Delft, supervisor
	Ir. E.G.M van Rhee,	Vanderlande supervisor
	Dr. ir. J.A.W.M. van Eekelen,	Vanderlande supervisor
	Dr. ir. Y.B. Eisma,	Member Examination Committee

This thesis is confidential and cannot be made public until December 19, 2025.

Automated Guide-Path Map Generation for Automated Guided Vehicle Systems

L.L. Endlich, *Student, TU Delft,* Dr. ir. J.C.F. de Winter, *Supervisor, TU Delft,* Ir. E.G.M. van Rhee, *Supervisor, Vanderlande,* and Dr. ir. J.A.W.M. van Eekelen, *Supervisor Vanderlande*

ABSTRACT

INTRODUCTION: The design of a guide-path map is a well-known problem in various industries, such as automated guided vehicle (AGV) systems, road networks, and train track design. Nowadays, designing a path map is a manual, time-consuming, and expensive process. The solutions often turn out to be far from optimal. Several approaches are known to generate path maps automatically. The main challenges are computation efficiency due to large solution spaces and the lack of feasibility of solutions for application to real-world layouts. Little research focuses on the importance of computational efficiency to keep up with the pace of dynamic user requirements and constraints.

OBJECTIVE: The goal is to design a framework that balances computational efficiency with feasibility in automated path map generation. A feasible path map can continuously distribute requested flow targets and is practically useable without many manual changes. It is expected that less optimal but still feasible solutions can be generated more efficiently by using a smaller solution space.

METHOD: The framework consists of four stages. First, the *practical problem* is converted into a *mathematical model*. A model that aims to resemble how humans generally could design path maps generates initial solutions, via exploratory or goal-oriented search, to decrease the solution space. The framework enhances initial solutions by adding paths in various ways such as minimizing the number of paths used and changing the layout decomposition. The variation in configuration introduces the ability to generate many different solutions that are validated on two aspects: capacity and performance. With linear programming, it is ensured that the requested flow can be distributed continuously. Each solution is ranked on criteria such as space utilization and redundancy. In addition, two experts are consulted to evaluate the solutions from a practical point of view. After validation, the stage of *mathematical solution* is reached. The final transition converts the modeled solution into a real-world path map containing paths over which AGVs could traverse (*practical solution*).

RESULTS: The framework is tested using three layouts: low complexity and small-scale, complex and large scale, and one in between. The exploratory model outperforms in the complex large-scale layout while the goal-oriented models outperform in the other layouts.

CONCLUSION: Path maps are generated in a reasonable and controllable time. The settings allow a trade-off between the efficiency and feasibility of solutions. Due to its efficient computation, the model has the potential to generate path maps that are directly usable in real applications, if several realistic implementations are included to improve the feasibility of solutions.

Practical Contributions: The path maps are useable for the exploratory stage of the current design process. Generated path maps can be used as starting solutions for engineers and after making some manual changes, the path maps are simulation-ready. The computational efficiency of the algorithm can ensure that the engineers can keep up with the rapidly changing user requests. Solutions are also suitable for the sales process to give customers validated possible path maps. When improving the framework, solutions could directly be used in simulations. This could contribute to a faster cheaper process of creating deliverables for customers.

I. Introduction

This paper proposes a novel algorithmic framework for automated guide-path map generation for automated guided vehicle (AGV) systems. AGV systems are a modern replacement for traditional conveyor systems in industrial warehouses, airports, and other applications in which transport and automation are used. With the rising trend of digitization in the spirit of Industry 4.0 (Hrušecká et al., 2019), the interest in AGV systems increases in manufacturing and assembly facilities (Arifin and Egbelu, 2000). An AGV system can be defined as an intelligent flexible automation system consisting of vehicles, a transportation network, an interface between a production or storage system, and the control system (Ganesharajah et al., 1998). Two types of AGVs are used in those systems: free-ranging and path-restricted unmanned vehicles (Mantel and Landeweerd, 1995). The first type uses equipment for position determination and collision avoidance, and the second type is more or less restricted to a fixed path structure, or a guide-path map. In this study, the latter type is considered. Consequently, the problem of designing path maps over which AGVs can traverse is introduced.

The problem of path map design does not concern AGV systems only. For example, consider the design of the road structure, how is it determined where and how highways should be built? Another example is the electricity distribution network: how should the network be designed such that all requirements are met? This paper connects the path map design problem to AGV systems.

In contradiction to conveyor systems, AGV systems do not necessarily need physically placed paths. Path maps can be designed as virtual guided path maps pre-programmed for the AGVs. Figure A.1 illustrates a virtually represented AGV complemented by a virtual AGV system shown in Figure A.2. AGV systems are mostly preferred over conveyor systems in automated production environments because of their routing flexibility, space utilization, safety, and reduction of overall costs (Reveliotis, 2000; Bilge and Tanchoco, 1997; Theunissen et al., 2018). Furthermore, AGV systems outperform traditional conveyor systems in terms of scalability and robustness (e.g. system failures) (Fransen and van Eekelen, 2021).

The design of a guide-path map for AGV systems directly influences the overall performance in terms of throughput, efficiency, and traffic management (Digani et al., 2014), see section A.I.1 for an example. In practice, the design path maps is a manual, time-consuming and expensive process (Beinschob et al., 2017). Therefore, it is a difficult challenge to keep up with the dynamic process of customer wishes and conditions that change over time (Ko and Egbelu, 2003). Besides that, the manual process often turns out to be far from optimal, especially when the sizes and complexities of environments increase (Digani et al., 2015). All in all, the manual design of path maps does not keep up with the pace of the booming AGV systems demand and requirements.

An advantage of the manual design of path maps is the human expertise and knowledge used. Experts with years of experience know how to design a feasible path map that is applicable in real-world environments. Combining the knowledge of human expertise with the ability to generate path maps in an acceptable amount of time would overcome the practical shortcomings in terms of flexibility to dynamic conditions.

Consequently, the research question of this study is defined as: *How to create an algorithmic framework that generates feasible path maps within an acceptable time?* More specifically, feasible refers to path maps that are applicable in real-world environments without a lot of manual changes or additions. Additionally, feasible path maps can continuously distribute the requested flow given certain requirements and constraints. Furthermore, an acceptable time means a generation time in the order of minutes for both simple and complex cases.

I.A State of the Art

Several algorithms for automated guide-path map generation are already present in the literature. Since path map design has a multiobjective nature, the solution space becomes large relatively quickly once the system contains many AGVs and flow targets (desired items per time unit from one point to another). For large solution spaces, exact optimization tools cannot be used since no solution can be found in a reasonable amount of time (De Ryck et al., 2020). An example of a relatively small environment and number of AGVs but large computation time is shown by Corréa et al., 2005. In this study, generating a path map for a 15x6 meter layout with 6 AGVs takes 10 minutes. Obviously, the computation time for large-scale environments up to 1000 agents (Li et al., 2020) will be too large and brings the advantage of automating the process in doubt.

An overview of existing strategies for tackling the guide-path layout problem is provided by Vis, 2006. Several mathematical optimization models have been designed to solve the problem. Gaskins and Tanchoco, 1987, created a model using zero-one programming that minimizes the total travel distance of loaded AGVs. Due to the mathematical complexity of the programming in combination with the large solution space, solutions are not found within an acceptable time. Furthermore, the return flow is not included in the model. Including the return flow is of great importance since knowing whether the requested flow can be sent toward certain locations is one part of the problem. The second part should be the validation of whether AGVs can keep doing that, AGVs have to return to some source (it validates the continuity of the flow distribution). Lastly, the zero-one programming model comes short in terms of flexibility because each time the input layout changes, a new program (both constraints and the objective function) should be formulated and solved.

An attempt to overcome the computationally inefficient zero-one programming is reported by Kaspi and Tanchoco, 1990. A follow-up model using the branch-and-bound procedure was created. Branch-and-bound algorithms search for the best solution for a given problem while using bounds to search parts of the solution space only implicitly (Clausen, 1999). In the branch-and-bound approach applied to AGV systems explained by Kaspi and Tanchoco, 1990, additional constraints related to return flow and connectivity constraints are included making the solutions more applicable to real-world environments. However, the algorithm is still not as efficient as planned.

Another follow-up model to overcome the computational efficiencies of both the zero-one programming and branch-and-bound is illustrated in Sinriech and Tanchoco, 1991. In this paper, the intersection graph method is introduced. The solution space in this method is decreased to gain efficiency by only using the intersection nodes rather than the entire network. Nevertheless, the intersection graph method is too computationally inefficient for large-scale problems as well. In Kiran et al., 1992, an integer programming formulation is used to solve the problem leading to optimal but, again, computationally expensive solutions. Integer programming is an optimization technique for maximizing or minimizing a linear function subject to various constraints. Important in integer programming is that all variables are restricted to be integers (Williams, 2009).

To overcome the increasing problem complexity resulting in a too large computation time, one could use or add a heuristic to decrease the solution space (Fazlollahtabar and Saidi-Mehrabad, 2015). An example algorithm combining traditional pathfinding with a heuristic to minimize computation time is presented in Uttendorf et al., 2016 and Uttendorf and Overmeyer, 2015. Besides significant efficiency improvements, the algorithm lacks several important aspects to perform well in terms of generating feasible path maps. Examples are the exclusion of the return flow, realistic turnarounds, and AGV-specific curve radiuses. The realistic turnarounds and AGV-specific curve radiuses that are missing are shown by the too-steep path corners and are a result of the solutions being too mathematical.

Another heuristic approach is illustrated in Goetz Jr and Egbelu, 1990, in which linear programming is formulated with a heuristic that decreases the solution space to major flow paths only. Linear programming is somehow similar to the previously discussed integer programming. However, decision variables now do not necessarily need to be integers and all requirements are represented by linear relationships.

Sinreich and Tanchoco, 1992, describe a heuristic approach as the optimal single-loop design method. The single-loop design method generates more realistic path maps compared to linear programming based on major flow paths described in Goetz Jr and Egbelu, 1990. What is included in the optimal single-loop method is the arrangement of flows resulting in flows more closely related to reality. Even though the computational efficiency has improved for both approaches, the generation time is still too large to keep up with the rapidly changing conditions. Additionally, when environments increase in complexity or size, the methods are not efficient enough.

None of the discussed methods above satisfy both the time efficiency and the feasible solution constraints mentioned in the research question. Either the problem that is optimized has a too large computation space, or the found solutions are not close to applicable to real-world environments.

What is missing in the literature is an approach that finds the balance between practical use (feasibility) and time efficiency. As explained, exact algorithms do not satisfy the time constraints. For some examples using heuristics, the time complexity is acceptable, but the solutions lack too much in terms of practical use and feasibility. This study fills this gap by providing a balance between computational efficiency and feasible solutions to overcome the previously mentioned shortcomings.

I.B Aim of the Present Study

As shown above, algorithms for automated path map generation are well known, but the combination of computation efficiency and feasible solutions is missing. This study focuses on designing an algorithmic framework that generates guide-path maps given static (no dynamic obstacles) environments while being computationally efficient, scalable, and meeting the required conditions. The path maps are constrained by unidirectional paths where AGVs only drive forward.

The aim of this study is not to find the most optimal route in terms of, for example, minimum path length or maximum possible throughput. A trade-off is made between practical applicability (feasibility) and time efficiency. Improving one aspect is expected to reduce the performance of the other. Therefore, this study is related to exploration research rather than exploitation research[1]. The loss of optimality or decreasing number of realistic constraints is less important than the expected gain in terms of computational efficiency. Resulting in the path map design process keeping up with dynamic changes in layout conditions, requirements, and constraints.

The proposed methodology uses a significantly different approach from those discussed in the literature. To improve the performance in terms of efficiency, the solution space must be decreased. In the literature, several approaches use optimization algorithms that compute what paths from all possibilities define the path map (large computation space). This study rather focuses on generating many different path maps followed by validation.

More specifically, the proposed algorithmic framework works as follows. First, multiple initial solutions are generated via a model that aims to resemble how humans generally could design path maps given a layout and points of interest. Then, the framework improves the initial solutions with additional bypasses, nearest points of interest connections, and other smart additions and rules. Since both the initial solution strategy and the implemented smart improvements can consist of multiple different configuration settings, many different

[1]Exploration refers to the discovery of new frameworks, exploitation is the refinement of existing frameworks (Sinha, 2015)

solutions can be generated. This introduces the possibility of ranking and selecting the best layout out of all.

Since the framework uses a model that imitates human path map designing choices when creating the solutions, proper validation is of great importance. Validation in the framework uses two aspects: capacity and performance validation. The capacity validation is computed via linear programming to investigate whether the requested flow can be distributed in a generated solution. The solution space changes from the entire environment to the generated solution, which means a significant decrease in solution space. Consequently, capacity validation is expected to be computationally efficient even if an optimization tool is used. Performance validation consists of two parts: a cost function and an expert analysis. The motivation for using both aspects is to analyse the differences between theoretical (cost function) and practical (expert analysis) performance. The cost function contains four criteria: path utilization, path intersections, occupancy rate, and redundancy. A more detailed description is provided in section II.C.2. The expert analysis, or evaluation, consists of interviews with two experts discussing the expected performance of multiple path maps generated by the proposed framework. Consult section III for a more detailed explanation of the in- and outputs of the expert analysis.

I.B.1) Proposed Methodology: The proposed methodology elaborates on generating solutions in various ways with a decreased solution space resulting in efficient computations. The proposed methodology consists of four stages, as shown in Figure 1 (consult section A.I.2 for a visualisation per stage). First, the practical problem is converted into a mathematical model as this creates the possibility of implementing both initial solution generation and smart mathematical improvements into the system. In this step, the practical 2D problem is converted into a graph with vertices and edges to introduce the previously discussed advantages of graphs to the model. Accordingly, an implemented mathematical algorithm analyzes the necessary conditions of the system. The mathematical validation is of great importance for this study because of the focus on computation efficiency over optimization and realistic constraint implementation. The final step transforms the mathematical solution into a practical real-world solution suitable for analysis and, if the quality of the solutions is sufficient, makes solutions usable for simulations.

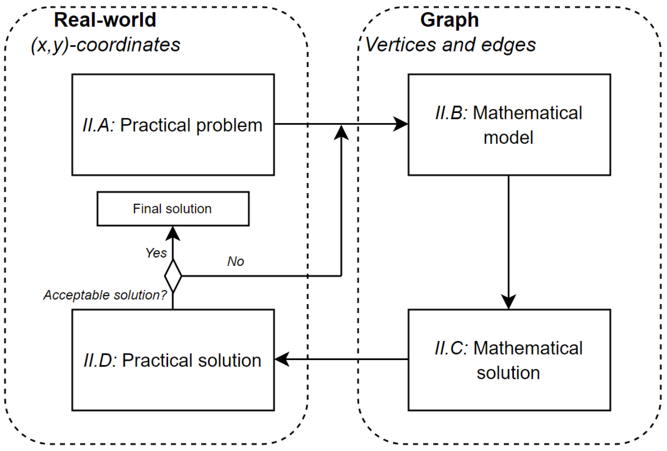


Fig. 1: Overview of the proposed methodology

In this study, path map generation and analyses for the proposed methodology are used. Two fictional (small- and large-scale) and one realistic environment (middle-scale) are used for comparison between the performance of different framework settings. The generation results of this study concerning the fictional layouts are compared to an exact optimization algorithm (Teusink, 2022). It is expected that due to the exact nature of the algorithm described in Teusink, 2022, more realistic and optimal layouts are generated while being less computationally efficient, compared to the proposed framework. Therefore, it is expected that the exact optimization algorithm outperforms the framework of this study for small-scale solutions in terms of feasibility as for small-scale layouts the solution space might be reasonable which could result in the quality of solutions outperforming the slightly larger computation times.

Considering the computational efficiency for larger layouts (large-scale), it is expected that the exact algorithm performs worse. The motivation for this comparison between frameworks is to analyse whether the hypothesis of gaining in terms of efficiency while losing some in terms of feasibility is valid. The comparison between the results of this study and the exact algorithm is made by analysing the performance validation for both models. To rank all simulations, an expert analysis will be executed to evaluate the solutions from a realistic point of view. Considering the aim of this study, it is expected that from this point of view some real-world applicable aspects are missing but without a lot of manual changes, the solutions could potentially become applicable or feasible to real-world environments.

The final results are obtained while testing the framework on a realistic layout from the industry. For this layout, no data is available, therefore, the expert analysis is more important than the performance computed by the cost function.

II. Algorithmic Framework

The proposed methodology is provided in more detail in the algorithmic framework in Figure B.1. The algorithm exists of five blocks: *Graph Generator*, *Path Finding*, *Capacity Check*, *Smoothening*, and *Validation*. After these steps, the practical problem is converted into a practical solution, as discussed in section I.B. In addition, it is shown in which state what part of each step takes place, either in 2D coordinates or in a graph. In the following subsections, each step is discussed in more detail, followed by definitions of the methods and algorithms used.

II.A Practical Problem

In the translation of a practical problem into a mathematical model, a 2D input environment is converted into a graph. An example input layout is illustrated in Figure 2 and explained in section B.II. Here, handling points (points of interest) refer to pickup (source), drop-off (destination), charging, or parking spots. Because of the shape of an AGV each handling point can be reached from two sides.

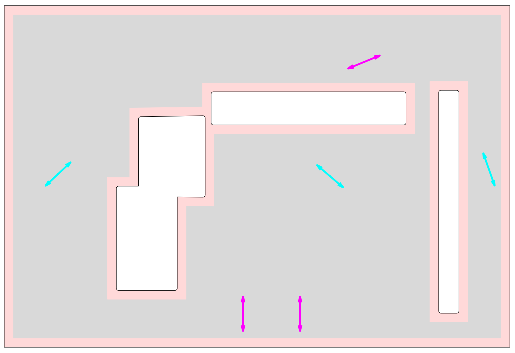


Fig. 2: Example fiction input layout

Next to an input layout, other real-world parameters could be incorporated into the practical problem. Examples are: handling points, vehicle characteristics (speed, dimensions), flow targets, battery information, minimum cornering angle, etc. For this study, the practical problem is defined as follows: *Given a layout with drivable space, handling point information, and vehicle size, a path map should be generated such that all flow targets are met without exceeding the maximum edge capacity.* Here, drivable space refers to the area in which AGVs can drive. Handling point information is important (coordinates, heading, and type) since the flow targets are defined from one handling point to one another, this is where the required flow is defined. Then, vehicle size gives an indication of how much safe space should be maintained between paths and obstacles. Lastly, the maximum edge capacity sets a limit on the maximum flow an edge can contain.

Returning to the proposed methodology visualised in section A.I.2, an example of a practical problem is shown at the top left as a layout with handling points and obstacles, with additional, information such as the flow targets and maximum edge capacity above it.

II.B Mathematical Model

The purpose of the mathematical model is to create initial solutions to decrease the solution space for the optimization and validation step. The initial solutions are generated via a model that resembles how humans generally could design path maps. To achieve this, the mathematical model is chosen as a directed graph. A graph representation of the environment is commonly used in robotics applications (i.e. path planning, localisation, and task allocation). Many robust methods have been developed that take advantage of the information from graphs (Urcola et al., 2015). Graphs are also used to represent layouts in path map design problems similar to this study (Liggett, 2000, Reith et al., 2021). A graph consists of vertices and edges, so information about relationships between data points is known, and knowledge or decisions can be implemented relatively easily and even used for validation.

Since a graph consists of fundamental units (vertices) connected by edges, the mathematical model consists of three key steps: vertex generation, edge generation, and choosing which edges will be used in what direction (direction assignment). The direction assignment step is where the imitation of humans drawing initial solutions takes place. Starting with a general explanation of graph theory, the strategies for all steps are discussed in this section. The visualisation of this step is shown in the layout at the top right in section A.I.2.

Graph Theory: An undirected graph is denoted by $\overline{G} = (V, E)$ in which V and E point out a set of vertices and edges, respectively. An advantage of graph theory is the indication of relationships in data. In the graph or network, vertices or nodes represent the entities (featureless and indivisible objects) and edges indicate the relationship between them. For undirected (or bidirectional) graphs, an edge $e \in E$ is defined as an unordered pair of vertices (u, v) where $u, v \in V$. Vertices u and v are called the endpoints of edge e. Note that for undirected graphs an edge (u, v) is equivalent to its opposite, (v, u).

Directed graphs are slightly different from undirected graphs and can be defined as $G = (V, E)$. Here, edges are defined as ordered pairs of vertices. This property gives directionality to each edge $e \in E$ such that $(u, v) \neq (v, u)$.

II.B.1) Vertex Generation: In the literature, vertices are generated using various methods. For example, in Reith et al., 2021, the vertices are generated at the corners of all static obstacles, Kavraki et al., 1996, use random vertex generation, and also often vertices in the form of an evenly spaced and divided grid are generated (Watanabe et al., 2001, Zhang et al., 2018). This study focuses primarily on both evenly divided and equally spaced (grid-like) and randomly placed vertex generation inside the drivable space. In addition, other implementations for vertex generation are introduced to complement the graph representation.

The strategy of the framework is to generate vertices V and edges E to generate a bidirectional, or undirected, graph $\overline{G}$. Handling points in the mathematical model are defined as (x, y, θ), a set of coordinates complemented by its heading. The first implementation for additional vertices is the generation of two new vertices in the extension of each handling point based on the heading. The motivation for this is to force a path through a handling point in the extension of its heading. Those vertices are called orientation handling points. An example of handling point η_i and the two generated orientation handling points η_{in} and η_{out} can be observed in Figure B.5. Besides the orientation handling vertices, no connections can be made to the original handling point vertices such that the correct heading always will be respected.

A second implementation is t generate more than two handling point orientation nodes. Adding more than two orientation handling points to a graph can mean the difference between a feasible and infeasible situation. An example is illustrated and explained in Figure B.6 and section B.III.1. A third option is to generate vertices at the intersections of lines through each of the handling orientation point nodes. This implementation creates the ability to generate vertices more smoothly around obstacles. For example, when two corridors intersect, this results in an additional vertex in the middle of the corridor intersection. Note that those variations already provide many opportunities for generating slightly different graphs that can eventually be turned into path maps.

The final step in vertex generation is removing duplicates and vertices too close to one another. The safety distance is equal to half the AGV size. The same size is used for the distance between evenly divided and equally spaced vertices. See Figure 3 for an example environment with vertices generated in blue, handling points coloured in purple and cyan, and the drivable space shown in grey. Here, the initially generated vertices are evenly divided and equally spaced. As shown, adjustments are made in the areas around handling points. For further explanation and examples consult section B.III.

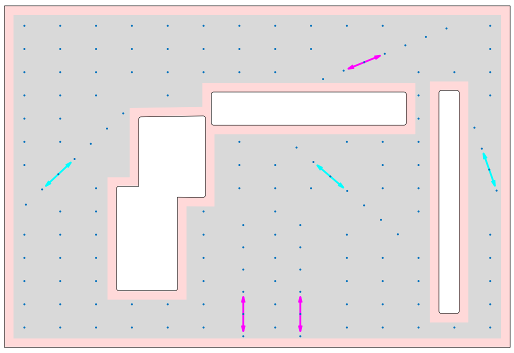


Fig. 3: Layout with generated vertices

II.B.2) Edge Generation: Graphs are generated by connecting the vertices with edges. The framework uses the strategy of generating a large number of potential edges, given a set of vertices. Then, from the large set of vertices, a selected group of edges is used to generate the actual graph. This is expected this significantly decrease the solution space resulting in increasing efficiency.

Examples of edge generation methods are Delaunay triangulation (Labatut et al., 2007), Voronoi diagram (Bhattacharya and Gavrilova, 2008) and medial axis theorem (Nieuwenhuisen et al., 2004).

Starting with the connection tool used in this research: Delaunay triangulation. The set of edges E for vertices V in 2D space is obtained by the following property: three vertices u, v, w are the vertices of a Delaunay triangle if and only if the sphere passing through the vertices does not contain any other vertex $x \in V$ in its interior (Devillers, 1999). An example of edges constructed via Delaunay triangulation is shown in Figure B.7.

In the literature, Delaunay triangulation and Voronoi diagrams often go hand in hand (i.e. Gallier, 2008; Fisher, 2004) because the Delaunay triangulation is the straight-line dual of the Voronoi diagram. See section B.IV.2 for elaboration on the Voronoi diagram.

Since the Voronoi diagram is generated using the set of vertices including the handling points, it is known that each handling point lies exactly at a Voronoi region center. Therefore, when using the connections between those centers (Delaunay triangulation) the advantage is introduced that all handling points are a vertex in the graph. For the same reason, the medial axis transform is not used for edge generation (Choi et al., 1997). The medial axis theorem uses the set of points that have two nearest points to the boundary of a shape and defines a skeleton of all points having more than one closest point to a boundary. Therefore, this method could be used if paths with maximum clearance distance are preferred. Both statements motivate the choice of Delaunay triangulation.

One drawback of Delaunay triangulation is that in some situations the solutions are not. In section B.IV.3, an example of a non-unique Delaunay triangulation is illustrated, followed by an explanation of the framework dealing with this issue. After removing all edges intersecting with obstacles or boundaries of the layout itself, an undirected graph is generated, as shown in Figure 4. For more information on edge generation, see section B.IV.

The third method is presented in Algorithm 3. Again, only connections from pickup to drop-off points, or the other way around, are made. Further explanation for this approach is illustrated in section B.V.2. An important aspect that should be validated here, is whether the directed graph is strongly connected. A directed graph is strongly connected if, for arbitrary vertices $u, v \in V$, there exists a directed path from u to v and from v to u (Frank, 1981). In other words, it should be possible to reach any node starting from any other node by traversing the directed paths. This is important in layouts for which each AGV should be able to visit all handling points. In section B.V.2 an example of both a strongly connected and a not connected graph is illustrated.

Figure 5 shows an example of a directed graph created by the loop generation method.

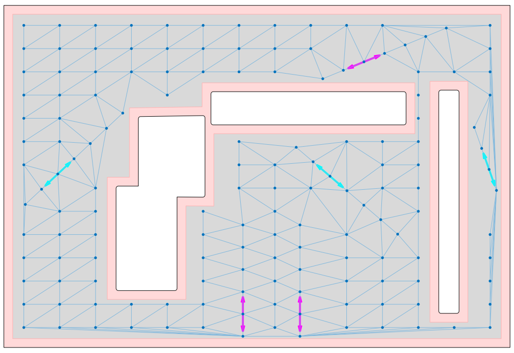


Fig. 4: Undirected graph in input layout

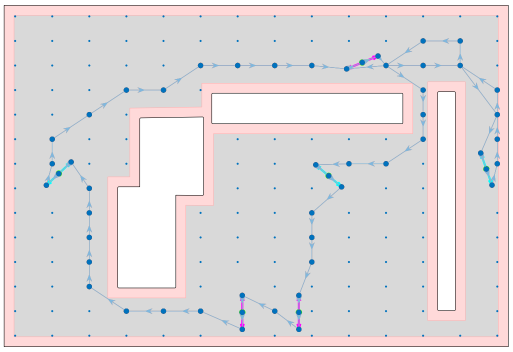


Fig. 5: Direction assignment applied to undirected graph

II.B.3) Direction Assignment: The undirected graph is transformed into a bidirectional graph over which (theoretically) flow could be sent through. Direction assignment can be done via, for example, mathematical optimization tools (Teusink, 2022), heuristics (Reith et al., 2021), or with the use of human-expertise-inspired models, for example, by traffic rule implementations and fuzzy-logic. As mentioned by Uttendorf and Overmeyer, 2015, a hybrid algorithm that combines human planning experience or expertise with a mathematical optimization process could result in producing efficient and applicable road maps for AGVs. Therefore, to initially generate directed paths, three solution strategies based on human-like path map design are incorporated into the algorithm. The strategies are found by manually drawing path maps given fifty layouts and handling points without any other conditions or constraints. From the manually drawn path, it was shown that the following three strategies were used in most solutions.

- Generating a Hamiltonian circuit[2] (a loop) through all handling points such that an AGV entering the path map can visit all handling points.
- Complementing the generated loop with bypasses between handling points to introduce additional paths for overtaking AGVs and find alternative routes in the path map.
- Connecting each pickup point to its k-closest drop-off point in the environment (and the other way around).

The first two strategies, or algorithms, have the same initial process of generating a loop through the handling points and orientation handling points. The order in which the handling points are connected is based on shortest distance. Starting at handling point η_i, the nearest $\eta \in H$ is connected. Accordingly, the same process for η_{i+1} is executed with η_i left out of H. If $H = \emptyset$ for η_j, it is connected to the starting point η_i to close the loop. Additionally, for the second implementation, bypasses from pickups to drop-offs or drop-offs to pickups are generated. See Figure B.12 and section B.V.1 for a visualisation and explanation. Additionally, Algorithm 1 and Algorithm 2 provide elaborations for each algorithm.

[2]Route in a directed graph from a beginning node to an ending node visiting each node exactly once (Garrod, 1966)

The directed graph is not applicable to real-world environments yet. The graphs show relationships between vertices and handling points and the possible vertices to reach from any point in the graph. The most important difference between the graph and a 2D representation of path maps is that the graph only shows what, but not how vertices can be reached.

The expectation for all three methods is divided into two categories: goal-oriented (or lookup) and exploratory search (Athukorala et al., 2016). The loop generator connects one handling point to one another. Complementary, the bypass generator adds target-specific routes. Therefore, all generated routes for the loop and bypass methods have a goal-oriented or navigational nature. For the k-nearest loop generator, the framework searches for the nearest handling points, as an exploratory search. It is expected that a goal-oriented search would outperform layouts without much drivable space (layouts with low complexity since not many options for alternative paths in drivable space are possible). For example, warehouses and airports with small corridors might block the exploratory search because there is not enough space for environmental acquisition. Moreover, in some cases, certain flow targets are of great importance in a layout (precise target goals). For example, in airports where baggage must be screened via a screening machine. All baggage pickups should ideally be connected without too many detours to the screening line. With goal-oriented search, those routes are expected to be more likely to be drawn.

Exploratory search is expected to be preferred in layouts with many small obstacles (more complex layouts) for which imprecise flow target goals have been established. Imprecise flow targets refer to the exclusion of differences in the importance of flow targets. An exploratory initial solution strategy could generate paths around those obstacles such that all handling points can be reached.

Dijkstra Shortest Path Algorithm: All proposed algorithms in this section use Dijkstra's single-source shortest path algorithm to connect the requested vertices in a graph (Dijkstra et al., 1959). In section B.VI, the principle of Dijkstra's algorithm is discussed and completed with an example.

An alternative method is the A* algorithm (Lester, 2005). The

A* algorithm uses a heuristic function to guide the search to find the optimal solution in a shorter period. For this study, A* could have been chosen since the mathematical model contains, for example, information about Euclidean distances. However, for ease of simplicity, it is chosen to use Dijkstra.

Configurations: The algorithmic framework can use several different configurations to generate many different path layouts, all based on slightly different strategies. A list of all configurations and parameters is presented in section B.VI.1. As discussed in the introduction, the validation process introduces the option of ranking the solutions based on performance. Therefore, advice for the most appropriate layout can be provided given the practical problem defined by the user. In section II.C.2, a more elaborated explanation of how each solution is validated in terms of performance is provided.

II.C Mathematical Solution

The validation consists of two parts: capacity and performance. Both validations take place in the mathematical solution. First, capacity validation is discussed, followed by performance validation.

II.C.1) Capacity Validation: Solutions are theoretically validated in terms of capacity according to the following strategy. First, the problem is formulated as a fractional multi-commodity flow problem (FMCFP). The FMCFP is then solved via linear programming such that the flow per edge distribution is computed while satisfying various constraints.

Fractional Multi-Commodity Flow Problem: In a multi-commodity flow problem (MCFP), multiple commodities move or flow from their respective source (origin vertex) to their sinks (destination vertex) without breaching any of the introduced constraints such as edge capacity, conservation of flow, or finding the cheapest path possible (Awerbuch and Leighton, 1994).

Let $G = (V, E)$ be a directed graph, where each edge $e \in E$ has a capacity $c(e)$. Consider a set of pick-up (source) and drop-off (sink) points defined by $s_1, ..., s_n \in V$ and $t_1, ..., t_m \in V$. There are k commodities[3], $k_1, ..., k_k \in K$. A commodity consists of three parts: the corresponding source and sink vertices and the requested flow demand. Consequently, a commodity is denoted as $k_i = (s_i, t_i, d_i)$, where s_i, t_i are the source and sink vertices and d_i defined as the demand, for example, in packages or suitcases per hour. Lastly, the fraction of flow i along edge e is expressed by $\psi_i(e)$. Hence, the flow of commodity i over edge e equals $\psi_i(e) \cdot d_i$.

MCFP either distributes commodities completely (denoted by 1) or not at all (denoted by 0). MCFP could be used for a new train structure or highway network with decisions about whether to connect two cities. For this study, the initial design of the directed graph could be solved as MCFP, for example, determining which edges to use in an undirected graph.

In FMCFP, commodities can be distributed by a fraction between 0 and 1. FMCFP can be used for this study since it is not a problem if the flow is distributed over edges, it is allowed to have a path map that is (almost) not in use. If pickup point p_i needs to deliver three suitcases to drop-off point d_j, given a maximum edge capacity, it is not a problem if two suitcases take route A and the third suitcase takes route B. Accordingly, the fraction of flow over an edge is defined as $\psi_i(e) \in [0, 1]$ instead of $\psi_i(e) \in \{0, 1\}$.

The advantage of FMCFP is that it can be solved in polynomial time through linear programming (Comen et al., 2009). In the flow assignment for this study, the objective function to be minimized equals zero because the only interesting outcome is whether it is possible to distribute the requested flow over all edges subject to the constraints.

[3]A commodity is defined as a good that must be transported from one or more source vertices to one or more sink vertices in the directed graph (Barnhart et al., 2001)

In the formulation of the FMCFP, multiple constraints need to be satisfied in the allocation for all flow variables. The first constraint validates whether the sum of all flows (of all commodities) over an edge does not exceed its maximum capacity. The flow per commodity over an edge is computed by multiplying the fraction of the commodity by its demand, as illustrated in Equation 1.

$$\forall (u, v) \in E : \sum_{i=1}^{k} \psi_i(u, v) \cdot d_i \leq c(u, v) \tag{1}$$

The second constraint satisfies the flow conservation for all vertices but handling points, where $u \in V_{\text{G}}$, and $V_{\text{G}} = V \backslash \{s_1, ..., s_n, t_1, ..., t_m\}$. Due to this constraint, no flow will be spawned (produced) or get lost at vertices, what flows in at vertex u should flow out, as shown in Equation 2. For ease of readability, it is determined not to divide d_i out of Equation 2.

$$\forall i \in K : \sum_{j \in V_{\text{G}}} \psi_i(u, j) \cdot d_i - \sum_{j \in V_{\text{G}}} \psi_i(j, u) \cdot d_i = 0 \tag{2}$$

Furthermore, two constraints for the flow conservation at source and sink vertices are illustrated by Equation 3 and Equation 4. The constraint of the flow conservation at the source implements that the desired flow needs to exit its source completely, even when it is split over multiple edges. In Equation 3, V_{P} refers to the set of pickup vertices, containing source vertices s_i.

The first component before the equality sign in Equation 3, refers to the flow leaving a sink. The second component ensures that no loops can exist such that flow leaving a sink enters itself without breaking the constraint. Therefore, what goes out minus what goes in should equal the demand for commodity i. In this study, the spawning or produced flow is defined as a negative demand, which means that at sink i a flow of d_i spawns.

$$\forall i \in K : \sum_{j \in V_{\text{P}}} \psi_i(s_i, j) \cdot d_i - \sum_{j \in V_{\text{P}}} \psi_i(j, s_i) \cdot d_i = d_i \tag{3}$$

For the incoming flow at sinks, or the place where flow gets lost, it is defined the other way around, as shown in Equation 4. In which V_{D} stands for the set of drop-off vertices.

$$\forall i \in K : \sum_{j \in V_{\text{D}}} \psi_i(j, s_i) \cdot d_i - \sum_{j \in V_{\text{D}}} \psi_i(s_i, j) \cdot d_i = d_i \tag{4}$$

Note that this formulation contains no auxiliary return flow-related constraints; it validates whether the flow can be distributed from sources to sinks. The return commodities do not necessarily contain one source and sink; for example, the return flow leaving a drop-off point may be distributed back to several pickup points. Consequently, each commodity needs its own return flow, leaving the sink it is sent to. For ease of simplicity, it is decided to include the return flow commodity as one commodity in the linear programming rather than as an additional commodity per commodity in the FMCFP. In the formulation of the linear programming problem, it is discussed how this is done in order to validate for auxiliary return flow as well in the validation step.

Linear Programming: By including fractional flows, the problem can be solved with linear programming in polynomial time. This satisfies the computational efficiency requirement. The solution for linear programming guarantees whether a directed graph can distribute the requested flow.

The goal of the linear programming method is to find vector $x_{e,k}$ that minimizes the linear objective function, as described in Equation 5b, subject to linear constraints defined in Equation 5c, Equation 5d, and Equation 5e. Vector f in the objective function refers to the coefficient vector, it defines how much each element $x \in x_{e,k}$ is weighted. The solution for the problem is vector $x_{e,k}$, for which $e \in E$ and $k \in K$. All $x_{i,j} \in x_{e,k}$ refer to the flow of commodity j on edge i. Therefore, the size of $x_{e,k}$ is equal to the number of edges times the number of commodities with the return commodity included.

In Equation 5, A and b refer to matrices and vectors, and l and u stand for constants. Further explanation on those parameters is provided later in this section.

$$\text{Find a vector} \quad x_{e,k} \tag{5a}$$

$$\text{that minimizes:} \quad f^T x_{e,k} \tag{5b}$$

$$\text{Subject to:} \quad A^{\text{ub}} x_{e,k} \leq b^{\text{ub}}, \tag{5c}$$

$$A^{\text{eq}} x_{e,k} = b^{\text{eq}}, \tag{5d}$$

$$l \leq x_{e,k} \leq u \tag{5e}$$

An advantage of linear programming is that the auxiliary return flow can be included more easily than in FMCFP. Since vector $x_{e,k}$ considers the flow per edge per commodity, the return commodity per vertex is included as well. This means that for each flow spawning at a source, the return commodity equals its opposite such that flow conservation holds. The same holds for incoming flow at sinks, all flows that disappear should spawn again such that the total system is in balance, and all individual return flows are combined into one additional commodity. See section B.VII for an illustration of the return flow given a graph and Equation 14.

Most of the matrices and vectors can be obtained directly from the FMCFP. In contradiction, coefficient vector f can be chosen to the user's wishes. In this study, three options are considered all leading to different results, as stated below.

1) f equal to vector filled with ones
2) f equal to vector filled with minus ones
3) f equal to vector filled with zeros

When choosing option one, the goal is to find the minimum flow per edge and the fewest amount of edges since the goal is to minimize the objective function. For the same minimization objective, the second option aims for the maximum flow per edge while using the largest number of edges. The third option has no unbounded coefficients per solution, all solutions $x_{i,j} \in x_{e,k}$ are randomly computed. Therefore, this option results in a possible solution; it does not matter what solution. Consequently, this improves the computational efficiency because fewer iterations are used. In this study, it does not matter how the flow is distributed. What is important is that it can be guaranteed that a feasible flow distribution exists. Note that this guarantee is theoretical and based on assumptions and parameters of the system. Therefore, it is not automatically guaranteed for real-world systems, but it likely is a necessary condition. For this study, the third option is chosen, see section B.VIII for further explanation of all three options for vector f.

Now that the objective function to minimize is determined, the three constraints will be further explained. Equation 5c is called the edge capacity constraint. It validates whether the inequality matrix A^{ub} multiplied with solution vector $x_{e,k}$, does not exceed the maximum allowed capacity per edge. This constraint sums the computed flow for each commodity per edge and then validates whether this is less or equal to the maximum capacity. The mathematical notation for this constraint can be observed from Equation 6. All $A^{\text{ub}}_{i,j}$ values contain the flow per commodity per edge, and all edge capacities $c(e)$ describe the maximum allowed edge capacity per edge e.

$$\begin{array}{c} \\ e_1 \\ e_2 \\ \vdots \\ e_e \end{array} \begin{array}{c} \begin{array}{cccc} x_{1,1} & x_{2,1} & \cdots & x_{e,k} \end{array} \\ \begin{bmatrix} A^{\text{ub}}_{1,1} & A^{\text{ub}}_{1,2} & \cdots & A^{\text{ub}}_{1,x_{e,k}} \\ A^{\text{ub}}_{2,1} & A^{\text{ub}}_{2,2} & \cdots & A^{\text{ub}}_{2,x_{e,k}} \\ \vdots & \vdots & \ddots & \vdots \\ A^{\text{ub}}_{e,1} & A^{\text{ub}}_{e,2} & \cdots & A^{\text{ub}}_{e,x_{e,k}} \end{bmatrix} \end{array} x_{e,k} \leq \begin{array}{c} b^{\text{ub}} \\ \begin{bmatrix} c(e_1) \\ c(e_2) \\ \vdots \\ c(e_e) \end{bmatrix} \end{array} \tag{6}$$

Equation 5d uses equality matrix A^{eq} and vector b^{eq}, and includes the flow conservation into the system. The matrix notation of this constraint can be observed from Equation 7. The rows of matrix A^{eq} iterate per commodity k over all vertices v in the directed graph. Then, each incoming and outgoing flow determines the columns of the matrix. The result of this multiplication is stored in vector b^{eq}, from b^{eq}_1 up to b^{eq}_{vk}. All parameters b^{eq}_i therefore refer to the incoming and outgoing flow, per vertex, per commodity. Since the columns iterate over each edge for each commodity, repeating matrices of information on each vertex and edge per commodity appear in A^{eq}, namely the incidence matrix. The incidence matrix represents the relations between the edges and vertices in a logical manner, a positive one is notated for incoming flow and a negative one for all outgoing flow (Van Nuffelen, 1976).

$$\begin{array}{c} \\ v_1, k_1 \\ v_2, k_1 \\ \vdots \\ v_v, k_k \end{array} \begin{array}{c} \begin{array}{cccc} x_{1,1} & x_{2,1} & \cdots & x_{e,k} \end{array} \\ \begin{bmatrix} A^{\text{eq}}_{1,1} & A^{\text{eq}}_{1,2} & \cdots & A^{\text{eq}}_{1,x_{e,k}} \\ A^{\text{eq}}_{2,1} & A^{\text{eq}}_{2,2} & \cdots & A^{\text{eq}}_{2,x_{e,k}} \\ \vdots & \vdots & \ddots & \vdots \\ A^{\text{eq}}_{vk,1} & A^{\text{eq}}_{vk,2} & \cdots & A^{\text{eq}}_{vk,x_{e,k}} \end{bmatrix} \end{array} x_{e,k} = \begin{array}{c} b^{\text{eq}} \\ \begin{bmatrix} b^{\text{eq}}_1 \\ b^{\text{eq}}_2 \\ \vdots \\ b^{\text{eq}}_{vk} \end{bmatrix} \end{array} \tag{7}$$

Lastly, Equation 5e includes boundaries on each solution $x_{i,j} \in x_{e,k}$. Note that this constraint differs from the edge capacity constraint as described in Equation 5c since each solution $x_{i,j}$ only describes the flow for an edge i per commodity j. Equation 5e ensures that each solution is less or equal to the maximum allowed edge capacity; therefore u is chosen as the edge with the least maximum allowed edge capacity in the directed graph. More important is the definition of the lower bound l. This part of the constraint secures that no negative solutions for $x_{i,j}$ are accepted, therefore l is set equal to zero such that the constraint is defined as $0 \leq x_{e,k} \leq \min(c(e))$.

Once the linear programming is formulated, it can be solved via, for example, dual-simplex and the interior-point algorithm. The dual simplex algorithm is an extension of Dantzig's simplex algorithm, in which linear inequalities are converted into equalities by introducing slack variables such that the constraints are transformed into solvable equalities with one definite answer (Nash, 2000). The proposed framework uses the dual simplex algorithm, consider section B.IX for further explanation and motivation.

Solution vector $x_{e,k}$ contains information about the distribution of flow in the directed graph. The slack variables provide information on how much fraction of the edge capacity is left on each solution $x_{i,j} \in x_{e,k}$ before constraints are exceeded. This is powerful information for the problem because one could analyse a graph and find occupancy information on each edge. This information can, for example, be used in decision-making to diverge or merge paths. Another advantage is that for infeasible solutions also slack variables are computed. This is valuable information for a path map because the slack variables provide information on each variable in the solution vector (all edges). Therefore, for each edge, it is known whether the constraints are satisfied or the model fails, so the weak spots where the path map results in failure are known.

II.C.2) Performance Validation: The performance validation consists of two parts: a cost function and an expert analysis. The criteria for the cost function were chosen after two individual interviews with experts complemented by criteria found in similar studies from the literature. Due to multiple simplifications, the cost function is a zero-order estimation only and not a guaranteed performance indicator. The outcomes of the cost function do not guide the algorithm toward certain decisions, the result is not used in a feedback loop.

The cost function is flexible, meaning that a user can emphasize or undervalue the cost components individually based on their own desires or importance. In the cost function, this is illustrated by the weights defined as $\lambda_1, \ldots, \lambda_4$. In Equation 8, the cost function is defined, consisting of four components. Here, U is the path utilization or footprint, I indicates the ratio between path intersections and the total number of path segments. Next, the occupancy per path segment is included by O, and lastly, R is the redundancy of the layout. All cost factors are normalized such that the dimensionless components $U, I, O, R \in [0, 1]$ can be summed. Note that in this validation, only capacity-wise feasible solutions are taken into consideration.

$$\text{Total cost } = \lambda_1 U + \lambda_2 I + \lambda_3 O + \lambda_4 R \tag{8}$$

Path Utilization: Equation 9 shows the definition of U, which refers to an indication of the amount of space used in the available space. In general, using less space while satisfying the given constraints results in preferable solutions. However, this can differ based on the use case and the customers' requirements. The used and possible spaces are computed in terms of edge lengths obtained from the generated graphs, it measures the footprint of the path map.

$$U = \frac{\text{Used edge length } [m]}{\text{Total possible edge length } [m]} \tag{9}$$

Path Intersections: In traffic layout design, four types of path intersections, or conflicts, exist: sequential, diverging, merging, and crossing (Thompson et al., 2009). Only merging and crossing conflicts are included in this study. Since all AGVs have relatively low vehicle speeds and equal speed limits, sequential conflicts are assumed not to be a problem. Diverging paths result in braking vehicles to slow down the speed such that directions can be changed. Again, due to the low vehicle speed, those types of conflicts are considered negligible.

Intersections are defined as vertices containing more than one incoming edge (merge) or more than one incoming and outgoing edge (crossing). Intersection component I computes the rate of average crossings per path, as illustrated in Equation 10. It is preferred to have fewer intersections in a layout because it often decreases the throughput. For ease of simplicity, it is assumed that more path intersections per path segment always negatively influence the overall path map performance.

Note that the negative influence on throughput is not always true. Consider path map $\mathcal{P}_m$ with 20 low-capacity crossings and path map $\mathcal{P}_f$ with one high-capacity cross. This would (unjustified) imply that cost($\mathcal{P}_f$) $>$ cost($\mathcal{P}_m$). But, again, the cost function is considered to be a zero-order estimation.

The intersections are counted in the graph phase of the algorithm, see section B.XI for an example and how this differs from counting in the geometrical phase.

$$I = \frac{\text{Number of intersections}[-]}{\text{Number of paths}[-]} \tag{10}$$

Occupancy Rate: The third component relates the average occupancy of a path segment to the maximum capacity, as presented in Equation 11. In general, a higher occupancy rate is preferred because then the path segment is used more efficiently. Therefore, the cost for the occupancy rate is computed as one minus the rate, a higher percentage lead to a lower cost.

$$O = 1 - \frac{\text{Average occupancy per path segment } [units/hour]}{\text{Maximum edge capacity } [units/hour]} \tag{11}$$

Redundancy: The final component is redundancy R. Redundancy is crucial for obtaining an efficient traffic flow (Digani et al., 2014) and can be explained using a well-formed infrastructure. A well-formed infrastructure has its endpoints distributed in such a way that any AGV standing on an endpoint does not completely obstruct other AGVs from moving between any other two endpoints (Čáp et al., 2015). Therefore, the generally assumed rule in this study is to have at least two substantially different paths leading toward each handling point.

R is determined by validating the number of different paths from each pickup point to each drop-off point, and the other way around. Consequently, the cost for this component equals zero if all empty AGVs have at least two alternative paths to each pickup point when leaving a drop-off location, and the same for each pickup point to each drop-off location.

To compute the entire contribution of redundancy to the cost function, all costs per pickup point $p \in P$ to each drop-off point $d \in D$ are computed as R_p. The costs from each drop-off point to each pickup point are defined as R_d, as presented in Equation 12a and Equation 12b. The number of possible different paths from a point v to u is denoted by $\rho(u, v)$. If the number of possible paths equals two or more, no costs are added for the corresponding point. For normalization, Equation 12c adds all costs per pickup and dropoff and divides this by the number of routes and back. Again, this is a simplification of the real-world influence of redundancy and therefore it provides a zero-order estimate only.

$$\forall p \in P, \forall d \in D : R_p = \begin{cases} 2 - \rho(p, d), & \text{if } \rho(p, d) \leq 2 \\ 0, & \text{otherwise} \end{cases} \tag{12a}$$

$$\forall p \in P, \forall d \in D : R_d = \begin{cases} 2 - \rho(d, p), & \text{if } \rho(d, p) \leq 2 \\ 0, & \text{otherwise} \end{cases} \tag{12b}$$

$$R = \frac{\sum_{p=1}^{P} R_p + \sum_{d=1}^{D} R_d}{D \cdot P \cdot 2} \tag{12c}$$

Weight Factors: With use of the weight factors in Equation 8, one can shape the cost function to a user's wishes. In industry, often a trade-off is made between a storage-driven and capacity-driven layout. Storage-driven could refer to a layout for which the path utilization should be as high as possible, for example, due to the lack of drivable space. In this case, λ_1 is defined as a large number such that more used path length is penalized more. But it could also work out the other way around: if there is enough drivable space that cannot be used for something else than paths, why not use it all? In this case, λ_1 could be close to zero, because it does not matter how the space is utilized.

In capacity-driven layouts, all that matters is a high-throughput system. Consequently, this means that the use of space is less important, and the occupancy rate and the path intersections are more important. If the occupancy rate is high, path segments are used efficiently, which means high flow traverses over path segments. Consequently, intersections will become high-capacity intersections. High-capacity crossings and merges are killing for overall throughput. Therefore, in this case, λ_2 and λ_3 are more important (higher weights) than to λ_1 and λ_4.

II.D Practical Solution

Once a mathematical solution is validated, the logistical view is converted back to a geometrical (real-world) view such that the generated paths are paths over which AGVs could drive, or at least in simulation. After the transition, the layouts contain information on how to reach one point from one another instead of connections between vertices only. In section A.I.2, the practical solution refers to the layout with the smooth path from one vertex to another, as shown in the left lower corner.

To convert directed graphs to geometrical 2D path maps, the proposed algorithmic framework makes use of Bézier curves for corner blending, this is called smoothing. An important advantage of Bézier curves is that they are computationally efficient (Shikhar Jaiswal, 2017). Complementary, Bézier curves are easy to manipulate because both the curves and their derivatives can be controlled directly, as shown by Chen and Huang, 2010. This results in the possibility of designing smooth curves automatically given a set of data. An example Bézier curve is illustrated in Figure B.24.

II.D.1) Bézier Curves: The smoothing phase of the proposed framework makes use of Quintic Bézier curves, which means it is a fifth-degree curve with six control points[4] denoted by $P_0, \ldots, P_5$, as presented in Figure 6.

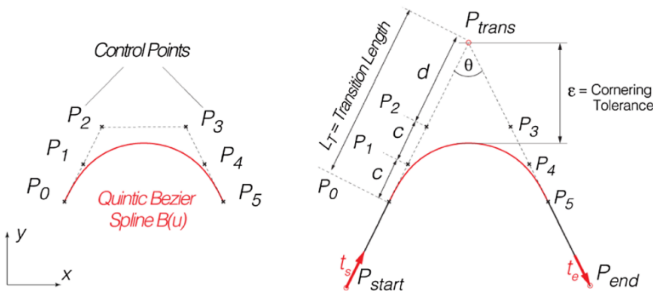


Fig. 6: Example of quintic Bézier smoothing by Adnan et al., 2020

Additionally, Figure 6 illustrates three corner points P_{start}, P_{trans}, and P_{end} that define the corner that needs to be smoothed. P_{start} is the starting point of the k^{th} edge (linear line segment in Figure 6). Then, the ending point of the k^{th} edge, and the starting point of the $(k+1)^{\text{th}}$ edge, is called the transition point P_{trans}. Consequently, P_{end} refers to the ending point of edge $(k+1)$ such that cornering angle θ is defined. Lastly, the Euclidean distance from P_0 to P_{trans} is defined as $2c+d$, which is required for further computing of control points. Consult section B.XII for the determination of all control points.

An important property for all curvatures $c \in C$ is that they are continuous. Curvature continuity is also called G^2, or in this study, smoothness between two adjacent Bézier curves (Lai and Ueng, 2001). G^2 continuity is important for path maps because due to the kinematics of AGVs; AGVs cannot drive over guide paths containing discontinuities. When making corner blends symmetric, concerning the angular bisectors of the two edges spanning the corner: $(e(P_{\text{start}}, P_{\text{trans}})$, and $e(P_{\text{trans}}, P_{\text{end}}))$, G^2 continuity is introduced to the Bézier curve connecting to their neighboring segments (Sencer and Shamoto, 2014). In section B.XIII it is illustrated how the symmetric property is achieved.

One more feature is added to complete the smoothing part of the proposed framework: corner point optimization. The red curves in Figure 7 visualise the converted path map given a directed graph G with vertices $v_1, \ldots, v_5$ and edges $e_1, \ldots, e_4$ colored in black. It can be observed that the path lengths of the corners are relatively large and the bend itself is rather gentle. A trade-off is be made between the steepness and the path length of the corner. In general, the maximum allowed speed at corners is lower compared to straight segments. On the other hand, the maximum allowed speed at a gentle corner is higher compared to a steep corner. Figure 8 illustrates the curve for the same corner points but now the curve exists of a small straight line segment in combination with a curve. An example of the consecutive corner problem together with the explanation of how curves as Figure 8 are generated, is illustrated in section B.XIV.

Another advantage of generating the corners as shown in Figure 8, is space utilisation. The curve uses less space in general but also results in less unused space remaining between the paths (around v_5). A downside of the smoothing is the unreachable vertex v_5, in this example. Nevertheless, this would only be a problem if this vertex would be a handling point, which can be solved by smart modeling of handling points. As discussed earlier, handling points are always modeled with at least two orientation vertices in the extension of their heading. This ensures that each handling point vertex is connected to two vertices with the same heading such that it is always located exactly on the path segments and not on a curve.

[4]An n^{th} order Bézier curve has $n+1$ control points to shape the corner (Shikhar Jaiswal, 2017)

Lastly, using the curves defined in Figure 8 tackles the problem of several consecutive corners. The curvatures are optimized such that the maximum curvature is minimized by estimating lengths c and d depending on angle θ. This allows AGVs to reach a higher cornering speed, while still staying within the acceleration limits of driving. The explanation is provided in section B.XV.

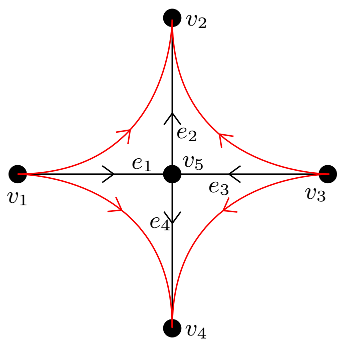


Fig. 7: Paths (red) based on graph

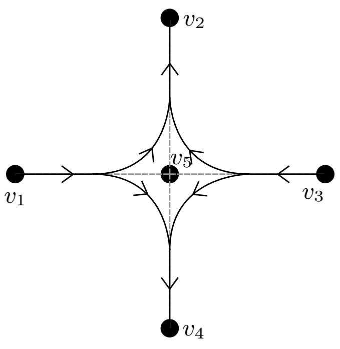


Fig. 8: New paths (black)

III. Results

The proposed framework generates solutions for three different input layouts: two fictitious and one realistic layout. As discussed in the introduction, a distinction is made between small- and large-scale and low and high-complex layouts. The fictitious layout results are compared to the automated path map generation solutions found in Teusink, 2022, which is called the exact algorithm in the rest of this study. Note that this algorithm slightly differs from this study, this is further explained in section III.A.4. The realistic layout solutions generated by the model of this study are validated by a path map used in the industry created by experts from Vanderlande[5]. The comparison between real and generated path maps is incorporated in the expert analysis, the visualisations of the real path maps cannot be used in this study due to the duty of confidentiality. All results are analysed using three criteria: computational efficiency, cost function, and expert analysis.

The expert analysis consists of feedback on path maps generated by the proposed algorithmic framework. The feedback was obtained by individually interviewing two experts with multiple years of experience in the process of manually designing path maps for AGV systems working at Vanderlande. Both experts were consulted to evaluate the solutions from a realistic point of view. In the interviews, the two best-performing solutions per layout (according to the cost function) were proposed and it was discussed what manual changes would be needed to use this layout in a realistic simulation. The expert analysis evaluates how close the solutions are to being suitable for simulations and emulations in terms of manual changes and therefore it reverts to the feasibility of the solutions. The outcome of the analysis is expressed in shortcomings and unrealistic behavior of generated path maps.

III.A Generation Experimental Layouts

First, the two fictional instances are explained followed by the generation setup including an elaboration on the weights used in the cost function. The two fictional instances are explained as follows:

- A small-scale and low complexity airport with a screening system.
- A large-scale high complexity parcel grid layout in which packages are picked up and delivered from one handling point to one another.

In section III.A.1 and section III.A.2, both environments are further explained in terms of goals determining the weights of the cost

[5]Vanderlande is a market-leading, global partner for future-proof logistic process automation in the warehousing, airports, and parcel sectors, https://www.vanderlande.com/about-vanderlande/.

function. Therefore, the cost function is personalized to the users and requirements. The motivation for using those two instances is supported by two arguments. First, the costs per layout can be compared to existing results from the literature that also made use of those layouts. Second, a clear distinction in layouts could validate the expectations between goal-oriented and exploratory configurations for the algorithmic framework.

III.A.1) Airport Instance: The layout of the airport instance is shown in Figure 9. Here, the drivable space, obstacles, pickup points, and drop-off points visualised in grey, white, cyan, and purple. The goal for the airport instance is to handle a requested capacity with a minimum amount of space in use. Since baggage must always be screened and sent back to passengers at the airport, there must always be a second option to traverse from a pickup point to a drop-off point in case an AGV is broken and blocks a path toward a pickup point. Therefore, redundancy is of great importance. Due to the lack of space, costs for path intersections are not emphasized. Furthermore, path occupancy is not that important because not too much baggage can enter the system. For further elaboration on the layout, see section C.I.1.

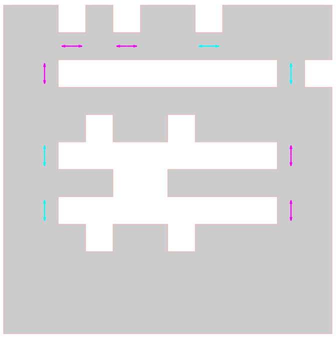


Fig. 9: Airport layout

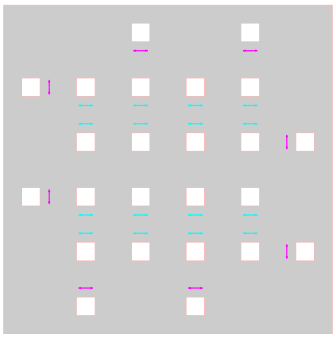


Fig. 10: Parcel layout

III.A.2) Parcel Grid Instance: An illustration of the parcel grid instance is shown in Figure 10. Compared to the airport instance, higher and more flow targets are required within many relatively small and symmetric corridors. The parcel grid instance is characterized as highly complex because of the large drivable space with many small obstacles such that the number of possible paths increases a lot. The main difference with the low-complexity airport layout is the options for multiple lanes paths in the corridors. In the airport layout, there is only space for one-lane paths in the corridors.

In the parcel layout, the least space possible is required in combination with a high occupancy rate. Therefore, avoiding path intersections is less important. Furthermore, redundancy is not of great importance for this instance since many bypasses, crossings, merges, and diverges are desirable such that AGVs can easily change lanes. Further explanation on the layout can be found in section C.I.2.

III.A.3) Generation Setup: The following generation settings are defined: flow targets, AGV length, maximum edge capacity, and cost function weights. The setup details for the fictional instances are provided in section C.I.3.

III.A.4) Solution Ranking: The solutions found by the proposed framework are compared to the solutions generated by the exact algorithm. Note that the input in terms of layout, flow targets, AGV size, edge capacity, and weights are identical for both frameworks. However, the frameworks themselves slightly differ on some grounds, consult section C.II for further elaboration on the differences.

III.B Results Airport Instance

The results of the airport instance for all generation setups can be found in section C.III. Complementary, the analysis performed by experts in path map generation is given in section C.V. The varying parameters in the setup are the number of vertices, the type of initial solution method (including varying k-neigbors and n-bypasses), and the setting that forces the algorithm to use the least amount of paths or not. The main findings are summarized below:

- Varying the number of generated vertices can transform narrow corridors from drivable to non-drivable space and the other way around. Considering the airport instance, it is the difference between a possible path between the two central obstacles.
- Forcing the framework to maximize the number of paths in use can transform infeasible path maps generated by the exploratory model settings into feasible ones because it adds alternative paths resulting in a strongly connected path map.
- The goal-oriented loop method is performing well according to the cost function, but from a practical point of view it is not suitable due to the lack of redundancy.
- Initial vertices generated as a grid always outperform randomly generated vertices for this instance.
- The worst performances originate from the k-loops generator.
- All solutions are generated in the order of minutes.

The best performing solution for this instance is found when generating a grid-like graph with 300 vertices, forcing the algorithm to generate the fewest possible paths, and applying the bypass method with n equal to two. The main contribution to the costs of this layout is redundancy. Several handling points can only be reached via one path without any alternatives as a result of the narrow corridors. For a comparison between the scores of the three best and worst performances, consult section C.III.5. Additionally, the computation time per solution can be found. The corresponding solution path map is shown in Figure 11.

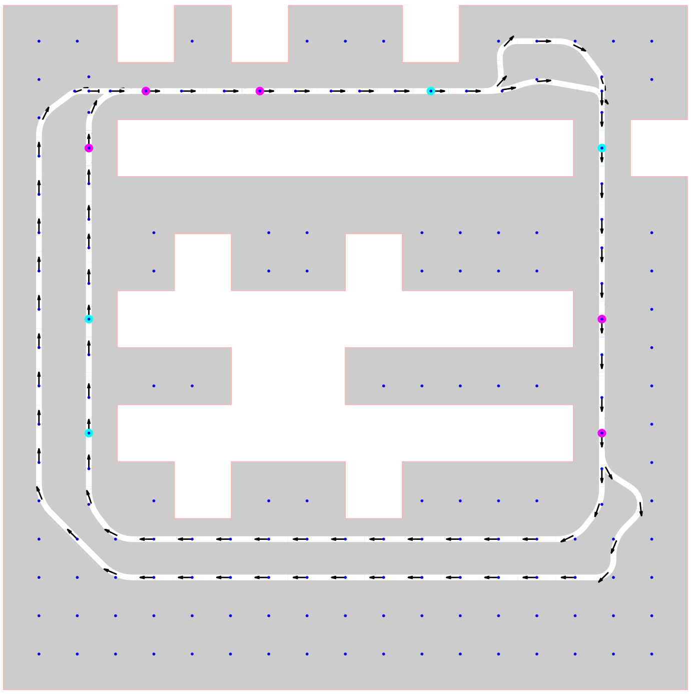


Fig. 11: Best performing solution for airport instance generations

The path is strongly connected such that all handling points can be visited. Additionally, an alternative path from the lower right corner to the upper left corner is generated to overtake AGVs if necessary.

A clear distinction between solutions being theoretically feasible and solutions being practically feasible is shown when comparing the cost function results and the expert analysis. As mentioned in the main findings, some solutions perform promising in terms of costs but would perform terribly in real-world applications, for example, the goal-oriented loop method. It was also mentioned that initially grid-like vertices always outperform the randomly generated ones. A reason for this is the lack of space in the small corridor in combination with the size of the environment itself. This means that are not many options for a path through this corridor and around the obstacles, often resulting in small detours or infeasible paths.

III.B.1) Comparison Exact Algorithm: The results for the exact algorithm are presented in section C.III.6; four corresponding path maps are generated in 849.57 seconds. Assuming that it takes the total time divided by the number of layouts for a solution to be found, one path map is generated in 212.3925 seconds.

All individual cost components and the computation for the best solution for both the proposed and the exact algorithm are shown in Table C.10. The results for the airport instance show that the exact algorithm outperforms the generation setups in terms of cost components. Moreover, the exact algorithm generates solutions closer to simulation-ready, according to the results of the expert analysis. The proposed framework shows unexplainable irregular path segments, a less smooth transition from a straight line segment to a curve, and unnecessary bypasses. However, the proposed framework results in terms of time complexity show significant improvements. Consequently, the expert analysis suggests that lots of potentially beneficial feasibility improvements could be made before the order of computation times approaches the exact algorithm.

Also, the results for the proposed framework are generated while varying only some of the possible parameters of this study. With slightly different parameters, entirely different layouts can be generated. For an example of parameters that could improve the generation setup, see section C.IV. A final notable difference in results between the proposed and exact frameworks is that the proposed framework has the ability to generate many slightly, or entirely different solutions. The exact algorithm does not include this in the model.

III.C Results Parcel Grid Instance

The generation setup for the parcel grid instance uses the following varying parameters: the heading of handling points, the number of nearest neighbors k, the number of bypasses n, and the decision for minimizing the number of paths used. Hence, the number of initially generated vertices is fixed. All results are discussed in section C.VI, and the expert analysis can be found in section C.VI.9. The main findings are:

- The loop generator can hardly generate path maps that can distribute the requested flow targets.
- The bypass generator in combination with minimizing the number of paths or ordering the handling points[6] is not appropriate for this instance. There are not enough diverging paths, therefore, to meet the flow targets of all handling points the maximum edge capacity is exceeded.
- The k-loops generator performs best in general, especially when minimizing the number of paths while ordering the handling points.
- Solutions for the k-loops generator turn feasible from $k \geq 3$ and costs increase while increasing k. The results show that those configuration settings generate enough paths to distribute the requested flow.

The best performing solution for the parcel instance, given the setup described in section C.VI, is obtained by generating multiple loops with k equal to three, in a randomly placed grid, while forcing the minimization of paths and ordering the headings of the handling points. The result is visualised in Figure 12 and shows loops through the two groups of clustering drop-off points. Attached to those loops are bypasses from, and to, pickup points, and from loop to loop. This structure of the path map makes it easy for AGVs to travel between pickups and drop-offs. The result also shows that the loops are not completely closed. This is not ideal since both gaps could force AGVs to take a detour their desired location.

[6]Ordering the handling points refers to choosing the headings such that a loop can be drawn through them without additional path segments or crossings when reaching the handling point form the correct angle.

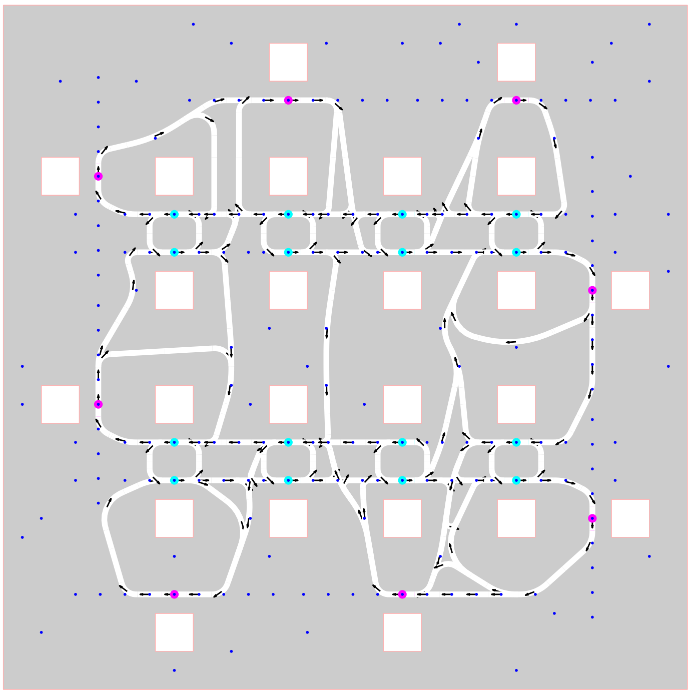


Fig. 12: Best performing solution for parcel instance generations

The three best and worst performances are summarized in section C.VI.6. In the best-performing solution, several irregular paths are included in the path map due to the randomly placed vertices. Another noticeable result is that the randomly placed vertices outperform the grid-like vertices in contrast to the airport instance. As this path map can meet the flow targets, enough diverging paths are generated to distribute the flow.

III.C.1) Comparison Exact Algorithm: The results are compared with the solutions found by the exact algorithm, as shown in section C.VI.7. Reasoning from the performance estimation of the zero-order cost function, the results show that the proposed framework outperforms the exact algorithm. In addition, the computation time is more than 800 times smaller. On the other hand, according to the expert analysis, the exact solutions are more realistic to real AGV systems.

Looking at the results and the cost components for all four criteria, the proposed algorithm is preferable only with respect to the occupancy costs. Larger path utilization costs result from the graph created by randomly placed vertices. Too many vertices are placed too close to one another, resulting in many vertices being removed. Therefore, less drivable space is actually used in graph generation. The solutions for the proposed and exact algorithm do not show too much difference in path length or space used, but the costs for the proposed algorithm are almost three times higher. This difference is explained by the definition of the path utilization costs, the total possible path length for the proposed framework solution is significantly smaller, if the used path length is similar, this results in higher costs.

In fact, the number of path segments and the total used path length for the proposed framework is smaller compared to the exact algorithm. Consequently, fewer paths are used for flow distribution, which increases the efficiency of the path segments and reduces occupancy costs. The difference in costs is emphasized by the weight factors. The results show, theoretically, feasible solutions given the generation setup. However, as elaborated in the expert analysis, manual changes are still needed to improve the feasibility before the solutions are suitable for real-world simulations. The last notable aspect shown in the results is that the exact framework generates only two solutions in the computation time. The proposed framework generates more than twenty theoretically feasible solutions in the order of minutes. This is a huge difference in performance of the framework.

III.D Generation Realistic Layouts

In section C.VII, the generation settings and results are presented for the realistic layout. The realistic layout contains a screening line (the straight long obstacle with two curves at the right center side of the figure) for which items are picked up, sent through, picked up again, and finally sent towards the exit conveyor (the obstacle located centrally in the layout). The items are picked up at the horizontal chain of pickup points at the bottom of the environment. Then, the items enter the screening line at the drop-off point in front and can be picked up at the pickup point at the exit of the screening line. The drop-off point for the items to exit the layout is located at the center in front of the exit conveyor. Several handling points in red refer to charging points, complemented by one parking spot in green. Lastly, the drop-off point at the left bottom of the layout is a clearance or manual encoding drop-off point. See Figure 13 for a visualization of the corresponding layout, a more detailed explanation is provided in section C.VII.

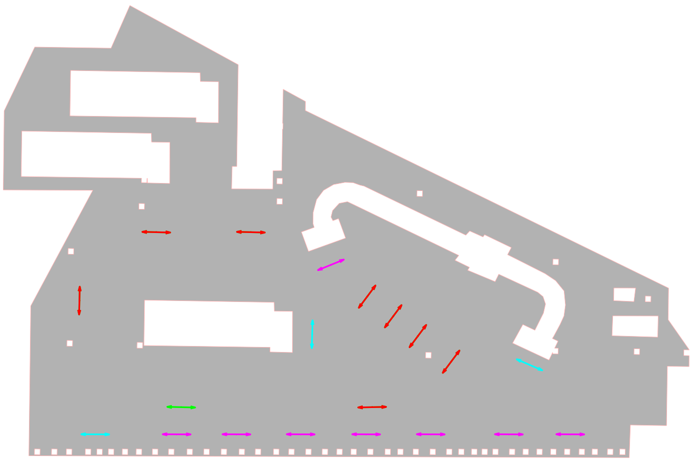


Fig. 13: Realistic layout

The used parameters are explained in section C.VII.1. There are no visualisations of path maps for comparison available due to confidential restrictions. Nevertheless, the cost function is used to discover its shortcomings when comparing the performance ranking to the results of the expert analysis, and to provide an indication of performance using different generation settings.

The varying parameters are minimizing the number of paths, the type of initial directed graph generation (loop, bypass, k-loops), the type of vertex generation (grid, random), and the number of bypasses and neighbors. The number of generated vertices and the headings of handling points are fixed. After analyzing the results presented in section C.VII, the following findings are obtained:

- The loop generator does not satisfy the redundancy requirements because there are hardly any alternative paths overtake the charging points in the layout.
- The bypass generator finds the best-performing solutions in terms of both the cost function and expert analysis. The results show that alternative paths for overtaking AGVs are included.
- The k-loops generator is not applicable for this instance because it requires too many path connections before creating a strongly connected graph.
- The best-performing path maps ranked from the realistic point of view differs from the zero-order estimation computed by the cost function.

As mentioned in the main findings, the outcome of the cost function analysis differs from the expert analysis. A reason for this is the zero-order approximation of the component computing the redundancy performance. The cost function does not include the importance of redundancy corresponding to the type of handling point. For example, it is more important that alternative paths are generated such that parked or charging AGVs can be overtaken compared to AGVs quickly picking or dropping items. The cost function does not distinguish those types of redundancy; it only calculates the number of possible different paths from each handling point to one another.

Another aspect missing in the redundancy component is information about alternative paths. For example, what type of handling points must be crossed and the length of an alternative path. In cases where the alternative path takes longer than waiting till an AGV is fully charged, or when an alternative route leads an AGV crossing another charging point does not contribute in a positive way to redundancy, in the equation it could.

One of the most important paths for this instance is the path from the drop-off location for the items towards the chain of pickup points that brings new items into the cycle. The best-performing layouts, according to the cost function, avoid direct path connection here, so it can only be reached via a huge detour. Theoretically, the algorithm says that the connection exists, but no path length or job completion time is taken into account. In the cost function, this is not a problem, but in practice, this would never work.

In Table C.21 and Table C.22, the three best and worst performing solutions according to the cost function are summarized. All least total costs solutions are simulated via the bypass generator, and the largest costs are via the k-loops generator. The reason why the latter has largers costs for the realistic instance is due to several clustering handling points in small areas. Therefore, before the outliers are connected from other handling points to reach a strongly connected path map, k needs to be increased a lot. If this is the case, the results show that too many paths are generated such that a huge percentage of the area is covered by paths resulting in large costs for both space utilization and path intersections.

The best-performing result for the realistic instance is chosen from the expert analysis perspective, as shown in Figure 14.

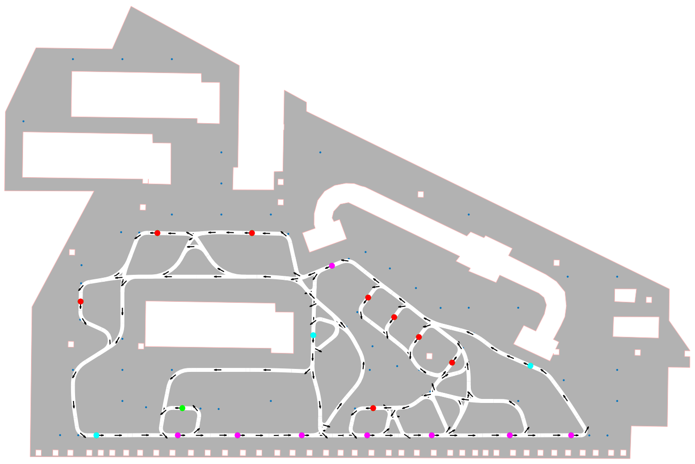


Fig. 14: Best performing solution for realistic instance

The result sufficient alternative paths such that all charging and parking spots can be surpassed easily when occupied. Additionally, one of the main connections leaving from the exit drop-off in the center of the layout towards the chain of pickup points at the bottom of the layout is included and even diverged such that different pickup points can be chosen to visit. Two drawbacks are the missing bypass to create the parallel path next to the drop-off point in the lower left corner and the bypass between the cluster of four charging points to the line segments containing seven pickup points.

The final observation about the results is the computation time. As the results show, a typical generation and validation time for solutions of the proposed framework is around five seconds. The real-world path map generated for this instance by engineers working at Vanderlande was designed and simulated in several weeks. For further analysis of this layout consult section C.VII.6.

IV. Discussion

A novel method for automated guide-path map generation for AGV systems was proposed in this study. The purpose of this method is to generate several feasible (theoretically capacity-wise and real-world applicable) solutions in a short period of time to improve the current, mostly manual, time-consuming, and expensive process of designing path maps. The proposed framework consists of three main steps. First, the practical problem is translated into a mathematical model in the form of a directed graph. Next, the mathematical solution is found in the form of capacity validation and score ranking. In the final step, the mathematical solution is converted back to a 2D geometrical solution for easier analysis.

This section first presents the main goal related to the research question. Then, the interpretations of the main results for all three layouts are reflected. The final subsection elaborates on multiple design choices and justifications for models and algorithms in the framework.

IV.A Research Goal

The results show that the proposed framework can generate many solutions within the required computation time in the order of minutes. Therefore, it can be concluded that this research complements the shortcomings in computational efficiency discussed in several studies (Corréa et al., 2005; Gaskins and Tanchoco, 1987; Teusink, 2022). The strength of this framework is that computation times, besides being reasonable, are also controllable. The largest contributor to computational time is capacity validation, which increases with the number of vertices and edges. Since those are inputs to the system, one could adjust those values to improve efficiency while decreasing solution quality.

Generating initial solutions with a model that resembles how humans would initially draw the path maps decreases the solution space enough for significant improvements in computation efficiency. Therefore, the recommendation of Uttendorf and Overmeyer, 2015, to combine human reasoning or knowledge with a mathematical optimization process is successfully incorporated into the proposed framework. The expert analyses confirm that solutions can potentially be improved in certain aspects from a realistic point of view. For further research concerning those aspects, one could consult the implemented fuzzy-logic rules presented in Uttendorf et al., 2016. Those rules were created through interviews with AGV system planners and include, for example, merging of paths, and path shortening if possible. Another study that may be promising in extension to this research is shown in Reith et al., 2021. Here, multiple implementations to improve existing lane design are outlined. Examples are the creation of new vertices to avoid path intersections and the exchange of lane parts in crowded areas in a layout.

Considering the feasibility, improvements are needed to generate simulation-ready solutions. A limitation of this study is that the capacity validation only provides a theoretical flow distribution. Once the solutions are simulation-ready, a throughput validation can be performed by simulating AGVs with task assignments, which gives an indication of actual performance. For example, in Digani et al., 2015, a comparison is performed between manually designed and automated generated path maps in terms of connectivity and redundancy, all based on simulation results. From here, claims reflecting on the simulation performances are made. For another example of a comparison of layouts using simulation data, one can consult Reith et al., 2021, or the comparison in throughput between manual and generated path maps in Uttendorf et al., 2016.

IV.B Results Interpretation

As earlier explained, a distinction between goal-oriented models (loop and bypass generators) and an exploratory search model (k-loops) is made. It was expected that the goal-oriented models would outperform for low-complexity and small-scale layouts whereas the exploratory model would be preferred for high-complexity and large-scale layouts.

Airport Instance: The distinction between the nature of models in the framework is also shown in the results. The goal-oriented configurations perform best for the low-complexity and small-scale airport instance. The initial starting solutions (Manhattan loop) for both goal-oriented methods, ensure that the path map is strongly connected. Since the flow targets are small compared to the high maximum edge capacity (factor of a hundred more), a strongly connected graph always leads to feasible solutions for this instance and settings. The differences between the loop and bypass solutions are the alternative paths that, most of the time, improve redundancy. On the other hand, this introduces the trade-off between space utilization (more paths need more space) and redundancy. Therefore, in some cases, the basic loop solutions outperform bypass solutions. If bypasses do not improve the redundancy of the overall system enough to compensate for the additional path costs.

The main issues for the exploratory k-nearest neighbors method in this layout are: creating path maps that are strongly connected and keeping the footprint small in the drivable space. To generate a strongly connected graph, the value for k needs to be too high so that more, and sometimes even all space, in the layout is used.

Parcel Grid Instance: In the parcel layout, there are many corridors, but the drivable space is relatively large with many drop-offs clustered together. Another important aspect of the parcel instance is the high number and high values for all flow targets. Consequently, the parcel instance is labeled as high-complex and large-scale. It was expected that for such instances, goal-oriented methods are not preferred. This is confirmed by the performance of the loop generator, almost all solutions are infeasible. This is consistent with the expectation since the defined flow targets equal 15 items per hour, per pickup point, to each of the 18 drop-off points. With an edge capacity of 600 items per hour, a Manhattan loop through all handling points would exceed the maximum capacity.

If enough bypasses result in diverging paths, the flow can theoretically be distributed. However, adding too many bypasses leads to too much space covered by the footprint. The results confirm this expectation. Therefore, it was expected that an exploratory search for the nearest handling points would outperform the goal-oriented methods. For this instance, it was expected that the best layout would exist of two inner loops through all drop-off points with all pickup points connected to them complemented with bypasses between the loops. This is also what the expert analyses and the cost function confirm.

Realistic Instance: The realistic layout is not exactly the same in complexity and scale as one of the fictional layouts. For this layout, the direction of important flow streams, in combination with redundancy at charging and parking spots are most important. It was expected that, given the specific important stream of flow, the goal-oriented methods would perform best. On the other hand, multiple clusters of handling points are located in the layout, which could be beneficial for the k-nearest neighbour method since once k is large enough to connect all clusters, the clusters are connected to each other.

The cost function ranks the bypass generator the best, as discussed above, corresponding to one aspect of the expectation. However, when the best-performing layout is considered from a realistic point of view, it is found that several important paths are missing to satisfy important flow targets. Thus, from a realistic point of view, the exploratory method outperforms the goal-oriented method.

Looking at the best-performing result for both the cost function and the realistic point of view, a combination of these would be ideal. The clusters of handling points could be connected via exploratory search, and the important stream of flow could be included via a goal-oriented path connection. This complements the shortcomings of both methods.

IV.C Reflection on the Algorithmic Framework

The second part of the research question relates to the feasibility of the path maps, which are analyzed by the validation. The validation consists of two components: a cost function and an expert analysis. The cost function provides a zero-order estimation of the performance from four perspectives: path utilization, path intersections, occupancy rate, and redundancy. Also, a theoretical capacity check is included by solving an FMCFP with linear programming such that a theoretical flow distribution is calculated. The second part of the validation presents an expert's point of view and reflects on solutions from a realistic perspective.

The results show that the proposed algorithmic framework is capable of generating computationally efficient path maps for both fictional and realistic environments. According to the expert analysis, unnecessary paths could be removed and sometimes essential paths are missing. However, even if manually designed path maps might look better and more logical, a sub-optimal, but efficiently generated path map might still be preferable if only a few changes have to be done manually (Uttendorf and Overmeyer, 2015). Therefore, it could be stated that, in terms of solution feasibility, this study is comparable to Uttendorf et al., 2016.

As the computation time is much shorter than expected, additional plug-ins to improve the feasibility of the solutions seem promising to the framework without losing too much efficiency. Improvements could be made in smoothness as well as in convergence to more realistic applicability. Smoothness could, for example, be improved by new, or slightly different implementations for graph generation and by introducing minimum edge cornering angles. A minimum curvature radius could be implemented while using Bézier curve smoothing, as explained in Digani et al., 2014, and Teusink, 2022. Another approach for improving the smoothness is by implementing additional features for vertex generation. One possible adaptation would be to generate vertices at the corner of all obstacles. This would introduce the possibility of smoothly cornering paths around obstacles. An example of this type of vertex generation is shown in Reith et al., 2021.

Real-world applicable improvements could be made in capacity validation. Path intersections or merges should be modeled differently from straight path segments since they reduce the maximum capacity. The same applies to situations in which paths are hindering one another, which affects the overall throughput. In Teusink, 2022, the hindering flow is modeled as a constrained maximum capacity. Edges hindering one another if the sweeps of AGVs are overlapping.

Parameter optimization could be implemented such that the best-performing solutions are selected for further investigation. Before the optimization can take place, significant improvements to the cost function are required. For example, the computation of the redundancy costs is too basic and unrealistic. The redundancy costs do not include information about alternative paths (e.g. path length, paths via parking and charging spots, etc.). In addition, the cost function does not include the importance of redundancy for certain handling points included. Some handling points, for example, a parking spot, do not necessarily need the ability of being reachable via multiple alternative paths. Digani et al., 2015, and Beinschob et al., 2017, describe a more realistic determination of redundancy in generated path maps.

Regarding the main findings in all simulation setups, different configurations (initial solution methods as well as different parameter settings) work better in specific instances. For now, the three initial solution methods mainly decide what the solutions will look like. This leaves enough space for further research to improve this part of initial solution generation. One could look into the initial generation of major flow paths, as shown in Goetz Jr and Egbelu, 1990. This confirms that the most important and largest flow targets are initially met, from here, additional paths could finalize the path maps. The same reflection holds for the parameters that finalize the initial solutions: there is much space for improvement. In addition to varying settings such as minimizing the number of paths, or changing the headings of handling points, several other types of parameters could be included.

The validation currently does not include, time, vehicle speed, and the number of AGVs required. If solution A has longer paths than solution B, and the frequency of items entering the system is constant, it means that more AGVs are needed. This can be proved by using Little's law (Little and Graves, 2008): $L = \lambda W$, for which L refers to the number of AGVs, λ equals the input of items per time unit, and W refers to the time an item stays in the system. If λ stays constant and W increases, the number of AGVs, L, needs to increase as well. More AGVs results in higher costs and different performances, so this is a limitation definitely worth mentioning before analysing generated path maps.

One of the advantages of linear programming is the set of slack variables containing valuable information on the directed graphs. As discussed earlier, the slack variables for feasible solutions contain information on where the maximum capacity of an edge is nearly exceeded. For infeasible solutions, they indicate where the system fails. Therefore, an important advantage of linear programming is not used in the current framework.

The proposed framework is a model-based approach, as new algorithmic manipulations are built to improve the current process of designing path maps. Due to the lack of data (existing public path layouts), it is not possible to target the problem discussed in this study with a data-driven approach. Data-driven approaches seem to be more applicable to dynamic environments while using sensor or camera data gathered by the AGVs. An example is shown in Gu et al., 2019. Using a different type of AGVs is a promising question, but it is a different type of problem compared to this study.

Practical Contributions: The practical contributions of this study consist of two parts: the exploratory or initial design stage of path map design by engineers, and the sales process. The computational efficiency perfectly matches the dynamic process of a customer seeking to implement an AGV system in an environment. Since many parameters can be manually tuned and varied in the framework, a large number of different solutions can be generated. This gives design engineers many different options as starting solutions and maybe even new insights into the process of path map design.

Regarding the sales process, the proposed framework can generate many different mathematically validated path maps in an exceptional generation time based on a user's input. Therefore, within an extremely short amount of time, the customer could see multiple different possibilities of path maps in the negotiation sales phase to explore possibilities in space.

IV.D Conclusions

A comprehensive algorithmic framework is provided that can be used for automated path map generation. The path maps are generated efficiently (in the order of minutes) and the solutions can be used in real-world applications without much manual modification. The computation efficiency requirements are met due to the decreased solution space before a mathematical optimization tool is used for validation. The main components of the framework are: converting the practical solution into a model (a graph), generating initial solutions with a model that mimics how humans might initially generate solutions complemented by specific parameters in the model, validating the solutions, converting the model solutions back to 2D path maps via smoothing.

With many variables in the configuration settings, the generation time is not only reasonable but also controllable. For example, using fewer vertices in layout decomposition decreases the computation complexity and solution quality, but it improves efficiency. Before the solutions can be generated simulation-ready, several real-world implementations need to be included in the model, such as a minimum cornering angle and influences of hindering paths. The validation of the solutions can also be improved. For example, by including vehicle speed, traveling time, and the number of AGVs needed. Nevertheless, the proposed framework is an appropriate solution to the exploratory research of automated path map generation. For now, the solutions could practically be used in the exploratory phase of path map design, or as a starting layout that can be improved manually until it is simulation-ready. A strategy, or the blueprint, for the automated and efficient generation of path maps is created in this study. From here, additional configurations or algorithms can be plugged in while using the same framework such that the solutions are tuned toward simulation applicability, or even toward personal wishes and requirements.

IV.E Recommendations

The framework consists of multiple complementary algorithms, therefore, several components for recommendations exist. The recommendations are elaborated separately for the different components.

IV.E.1) Mathematical Model: Evaluating the methods used for vertex generation, several improvements or additions can be made, as explained in the discussion. By adding vertices around obstacles, at the boundaries of the drivable space, and around points of interest, cleaner but also shorter and tighter path maps can be generated. The additional vertices also allow edges to be generated in all locations in the drivable space, for example in small corridors. Furthermore, the randomly generated vertices often perform worse than grid-like vertices. Therefore, randomly generated vertices could be used as a backup configuration when no feasible solution for a grid-like generated graph is found. It could also be applied to generate vertices at areas of the drivable space, or in addition to the graph to create a denser grid.

Regarding edge generation, it would be beneficial to already validate paths too close to one another (hindering paths) within the graphs. Note that this can also be done in after the paths are smoothed, but it could already make a difference in direction assignment. Additional methods for initial solution generation could also be used. For example, initially assigning paths based on the highest flow targets or priority. Another option is to combine several methods, one could use k-loops at clustered handling points and the bypass generator for connecting the loops, for example.

IV.E.2) Mathematical Solution: The advantage of linear programming consists of two parts: computation time and solution information. The linear programming solver finds a solution within polynomial time. In addition, the outcome of solving linear programming contains a lot of valuable information that is currently unused in the framework. When a feasible solution is found, the slack variables in the solution show values for each edge in terms of occupancy rate. This information could be visualised in a heat map to see where diverging of path segments might be useful before running simulations. Similarly, considering infeasible solutions, the slack values explain where the edges failing are. Again, this helps the user to find the weak spots of a path map such that it can be improved.

The current validation does not guide the path generation. If the framework uses validation results in a feedback loop, the framework could tune the solutions for better performance. One way of doing this is by using the slack variables, as explained earlier. Another method is to use the cost function results. However, if this implementation would be added, the cost function should be improved such that it not only provides a zero-order estimation of the cost criteria but a more realistic indication of performance. Also, some improvements for the linear programming problem could be added. For example, the maximum allowed edge capacity at merges and crossings could be estimated instead of using the maximum straight-line capacity.

The redundancy computation used in the cost function is too basic. It is recommended to improve this, for example, by constantly picking edges to remove from the directed graph and validating whether it is still possible to reach all handling points. Also, weights could be added for the redundancy of certain handling points. It could be more important for a parking spot to have multiple in and outgoing paths compared to a pickup point, assuming that picking up an item is a quick process. Also, weights could be added for the redundancy of certain handling points to assign importance to each handling point. This could be beneficial for a parking spot to have multiple in and outgoing paths compared to a pickup point, assuming that picking up an item is a fast process.

As mentioned in the discussion, the inclusion of (alternative) path length information, the number of AGVs needed, and vehicle speed are not included in the framework. Vehicle speed can drastically influence the throughput if too many diverges, merges, and crossings exist since AGVs would constantly need to slow down and accelerate here.

Considering the path utilization cost component, currently, this is computed by dividing the total edge length by the total possible path length. This definition does not include the vehicle sweep or space that cannot be used in between paths. It is recommended to use the used space including vehicle sweep around paths ($[m^2]$), divided by the drivable space ($[m^2]$).

IV.E.3) Practical Solution: The solutions are not ready for real-world simulations yet. Parameters recommended to include in the algorithmic framework are minimum corner angle and distance, the influence of hindering paths, and the removal of unnecessary paths. For example, a path from one pickup point to one another often does not make sense and could thus be removed. Before running simulations, realistic flow targets for a real layout can be tested with the current validation process to get an indication of the performance of the linear programming flow distribution calculations.

Furthermore, smoothing can be applied between multiple path segments to filter out irregular stairs-like path segments in a layout. One way of doing this is to replace several one-lane Bézier curves with one smooth curve, if possible due to drivable space constraints. Furthermore, it is recommended to have paths merging and diverging before and after handling points at a larger distance. This ensures that AGVs have sufficient space and time to enter the handling point from the correct angle.

The expert analysis validates solutions from a practical point of view by providing an indication of the expected performance. The expert analysis consists of only two experts. Therefore, a limitation of this validation is that the outcome of interviewing more than two experts could be different. It is recommended to interview a larger number of experts and incorporate their feedback into the framework. For example, hindering paths or too many crossings on a small area could be processed in the model generation already, the expert analysis could be used as human-algorithm complementary validation rather than human-only validation.

IV.E.4) Algorithmic Framework: Recommendations for the framework itself are to use the validation results to guide toward desirable solutions, as discussed earlier. The advantage of using those cost components is that the user determines what is important in the layout, so the algorithm would then tune itself based on the input. The final recommendation is to optimize the parameters within the different configurations based on an improved cost function such that the best-performing solutions, based on the user's criteria, are ranked correctly. All configurations used for the generations in this study are not close to all possible configurations. Therefore, it cannot be stated that the best-performing solutions have already been found.

Acknowledgments

With an accomplished feeling, I present my thesis to you, I sincerely hope you will enjoy reading it. I would like to express my appreciation to my supervisors, TU Delft, and Vanderlande. Besides each and every aspect of this process giving me unique insights as a researcher, also life at a company has been introduced. To top it all off, Vanderlande gave me the opportunity to provide a lunch lecture at TU Delft in collaboration with the student association of my very own studies: Asimov. Thank you all, and as said by a very famous man named Einstein (and my supervisors):

If you can't explain it to a six-year-old, you don't understand it yourself.

And I picked up my pen, and began to draw path maps on paper again, before starting programming the proposed framework like a headless chicken.

Code Archive: Due to company confidentially restrictions the code archive cannot be provided. The automated vehicles algorithms team of Vanderlande will continue working with the created code for this study.

References

Adnan, S. B. Z., Ariffin, A. A. M., & Misro, M. Y. (2020). Curve fitting using quintic trigonometric bézier curve. *AIP Conference Proceedings*, *2266*(1), 040009.

Andersen, E. D., & Andersen, K. D. (2000). The mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm. In *High performance optimization* (pp. 197–232). Springer.

Ari Muzakir, H. (2020). Bellman-ford algorithm for completion of route determination: An experimental study. *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)*, *6*(1), 29–35.

Arifin, R., & Egbelu, P. J. (2000). Determination of vehicle requirements in automated guided vehicle systems: A statistical approach. *Production Planning & Control*, *11*(3), 258–270.

Athukorala, K., Głowacka, D., Jacucci, G., Oulasvirta, A., & Vreeken, J. (2016). Is exploratory search different? a comparison of information search behavior for exploratory and lookup tasks. *Journal of the Association for Information Science and Technology*, *67*(11), 2635–2651.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, *23*(3), 345–405.

Awerbuch, B., & Leighton, T. (1994). Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, 487–496.

Barnhart, C., Krishnan, N., & Vance, P. H. (2001). Multicommodity flow problemsmulticommodity flow problems. In C. A. Floudas & P. M. Pardalos (Eds.), *Encyclopedia of optimization* (pp. 1583–1591). Springer US. https://doi.org/10.1007/0-306-48332-7_316

Beinschob, P., Meyer, M., Reinke, C., Digani, V., Secchi, C., & Sabattini, L. (2017). Semi-automated map creation for fast deployment of agv fleets in modern logistics. *Robotics and Autonomous Systems*, *87*, 281–295.

Bhattacharya, P., & Gavrilova, M. L. (2008). Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path. *IEEE Robotics & Automation Magazine*, *15*(2), 58–66.

Bilge, Ü., & Tanchoco, J. M. (1997). Agv systems with multi-load carriers: Basic issues and potential benefits. *Journal of Manufacturing Systems*, *16*(3), 159–174.

Čáp, M., Vokřínek, J., & Kleiner, A. (2015). Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. *Proceedings of the international conference on automated planning and scheduling*, *25*, 324–332.

Chen, E. Y., & Huang, M. (2010). Using bézier curve to improve the accuracy in integrated circuit design analysis.

Choi, H. I., Choi, S. W., & Moon, H. P. (1997). Mathematical theory of medial axis transform. *pacific journal of mathematics*, *181*(1), 57–88.

Clausen, J. (1999). Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, 1–30.

Comen, T. H., Leiserson, C. E., Rivest, R. L., & Clifford, S. (2009). *Introduction to algorithms*. MIT press.

Corréa, A. I., Langevin, A., & Rousseau, L.-M. (2005). A scheduling and conflict-free routing problem solved with a hybrid constraint programming/mixed integer programming approach. *Les Cahiers du GERAD ISSN*, *711*, 2440.

De Ryck, M., Versteyhe, M., & Debrouwere, F. (2020). Automated guided vehicle systems, state-of-the-art control algorithms and techniques. *Journal of Manufacturing Systems*, *54*, 152–173.

Devillers, O. (1999). On deletion in delaunay triangulations. *Proceedings of the fifteenth annual symposium on Computational geometry*, 181–188.

Digani, V., Sabattini, L., Secchi, C., & Fantuzzi, C. (2014). An automatic approach for the generation of the roadmap for multi-agv systems in an industrial environment. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1736–1741.

Digani, V., Sabattini, L., Secchi, C., & Fantuzzi, C. (2015). Ensemble coordination approach in multi-agv systems applied to industrial warehouses. *IEEE Transactions on Automation Science and Engineering*, *12*(3), 922–934.

Dijkstra, E. W., et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271.

Fazlollahtabar, H., & Saidi-Mehrabad, M. (2015). Methodologies to optimize automated guided vehicle scheduling and routing problems: A review study. *Journal of Intelligent & Robotic Systems*, *77*(3), 525–545.

Fisher, J. (2004). Visualizing the connection among convex hull, voronoi diagram and delaunay triangulation. *37th Midwest instruction and computing symposium*.

Frank, A. (1981). How to make a digraph strongly connected. *Combinatorica*, *1*(2), 145–153.

Fransen, K., & van Eekelen, J. (2021). Efficient path planning for automated guided vehicles using a*(astar) algorithm incorporating turning costs in search heuristic. *International Journal of Production Research*, 1–19.

Gallier, J. (2008). Notes on convex sets, polytopes, polyhedra, combinatorial topology, voronoi diagrams and delaunay triangulations. *arXiv preprint arXiv:0805.0292*.

Ganesharajah, T., Hall, N. G., & Sriskandarajah, C. (1998). Design and operational issues in agv-served manufacturing systems. *Annals of operations Research*, *76*, 109–154.

Garrod, C. (1966). Hamiltonian path-integral methods. *Reviews of Modern Physics*, *38*(3), 483.

Gaskins, R. J., & Tanchoco, J. M. (1987). Flow path design for automated guided vehicle systems. *International Journal of Production Research*, *25*(5), 667–676.

Goetz Jr, W. G., & Egbelu, P. J. (1990). Guide path design and location of load pick-up/drop-off points for an automated guided vehicle system. *The International Journal of Production Research*, *28*(5), 927–941.

Gu, G., Hong, Z., & Luo, D. (2019). A data-driven intelligent algorithm for dynamic path design of automated-guided vehicle systems. *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 1106–1111.

Hrušecká, D., Lopes, R. B., & Juřičková, E. (2019). Challenges in the introduction of agvs in production lines: Case studies in the automotive industry. *Serbian Journal of Management*.

Huerta, F. Z., & de Lara, R. G. L. (2015). Implementation of alternative solutions in linear programming modeling using the dual simplex method and duality method from primal problem, establishing implementation through the simplex methodology. *Global Journals of Research in Engineering*, *15*(1), 21–28.

Kaspi, M., & Tanchoco, J. (1990). Optimal flow path design of unidirectional agv systems. *The International Journal of Production Research*, *28*(6), 1023–1030.

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, *12*(4), 566–580.

Kiran, A. S., Unal, A. T., & Karabati, S. (1992). A location problem on unicyclic networks: Balanced case. *European Journal of Operational Research*, *62*(2), 194–202.

Ko, K.-C., & Egbelu, P. (2003). Unidirectional agv guidepath network design: A heuristic algorithm. *International Journal of Production Research*, *41*(10), 2325–2343.

Koberstein, A. (2005). The dual simplex method, techniques for a fast and stable implementation. *Unpublished doctoral thesis, Universität Paderborn, Paderborn, Germany*.

Labatut, P., Pons, J.-P., & Keriven, R. (2007). Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. *2007 IEEE 11th international conference on computer vision*, 1–8.

Lai, J.-Y., & Ueng, W.-D. (2001). G2 continuity for multiple surfaces fitting. *The International Journal of Advanced Manufacturing Technology*, *17*(8), 575–585.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Lester, P. (2005). A* pathfinding for beginners. *online]. GameDev WebSite. http://www. gamedev. net/reference/articles/article2003. asp (Acesso em 08/02/2009)*.

Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. S., & Koenig, S. (2020). Lifelong multi-agent path finding in large-scale warehouses. *AAMAS*, 1898–1900.

Liggett, R. S. (2000). Automated facilities layout: Past, present and future. *Automation in construction*, *9*(2), 197–215.

Little, J. D., & Graves, S. C. (2008). Little's law. In *Building intuition* (pp. 81–100). Springer.

Mantel, R. J., & Landeweerd, H. R. (1995). Design and operational control of an agv system. *International Journal of Production Economics*, *41*(1-3), 257–266.

Nash, J. (2000). The (dantzig) simplex method for linear programming. *Computing in Science & Engineering*, *2*(1), 29–31. https://doi.org/10.1109/5992.814654

Nieuwenhuisen, D., Kamphuis, A., Mooijekind, M., & Overmars, M. H. (2004). Automatic construction of roadmaps for path planning in games. *International Conference on Computer Games: Artificial Intelligence, Design and Education*, 285–292.

Reith, K.-B., Rank, S., & Schmidt, T. (2021). Conflict-minimal routing for free-ranging transportation vehicles in in-house logistics based on an a-priori lane design. *Journal of Manufacturing Systems*, *61*, 97–111.

Reveliotis, S. A. (2000). Conflict resolution in agv systems. *Iie Transactions*, *32*(7), 647–659.

Robere, R. (2012). Interior point methods and linear programming. *University of Toronto*, 1–15.

Sainlot, M., Nivoliers, V., & Attali, D. (2017). Restricting voronoi diagrams to meshes using corner validation. *Computer Graphics Forum*, *36*(5), 81–91.

Sampaio, J. H. (2017). Designing three-dimensional directional well trajectories using bézier curves. *Journal of Energy Resources Technology*, *139*(3).

Sencer, B., & Shamoto, E. (2014). Curvature-continuous sharp corner smoothing scheme for cartesian motion systems. *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*, 374–379.

Shikhar Jaiswal, A. (2017). Shape parameterization of airfoil shapes using bezier curves. In *Innovative design and development practices in aerospace and automotive engineering* (pp. 79–85). Springer.

Sinha, S. (2015). The exploration–exploitation dilemma: A review in the context of managing growth of new ventures. *Vikalpa*, *40*(3), 313–323.

Sinreich, D., & Tanchoco, J. (1992). The centroid projection method for locating pick-up and delivery stations in single-loop agv systems. *Journal of Manufacturing Systems*, *11*(4), 297–307.

Sinriech, D., & Tanchoco, J. M. A. (1991). Intersection graph method for agv flow path design. *The International Journal of Production Research*, *29*(9), 1725–1732.

Teusink, J. (2022). Automated path map generation for agv systems.

Theunissen, J., Xu, H., Zhong, R. Y., & Xu, X. (2018). Smart agv system for manufacturing shopfloor in the context of industry 4.0. *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, 1–6.

Thompson, M. K., Park, M. J., Kwon, Q., Ibragimova, E., Lee, H., & Muyng, S. (2009). The application of axiomatic design theory and conflict techniques for the design of intersections: Part 2. *Proceedings of the Fifth International Conference on Axiomatic Design*, 129–136.

Urcola, P., Lazaro, M., Castellanos, J., & Montano, L. (2015). Generation of probabilistic graphs for path planning from stochastic maps. *2015 European Conference on Mobile Robots (ECMR)*, 1–7.

Uttendorf, S., Eilert, B., & Overmeyer, L. (2016). A fuzzy logic expert system for the automated generation of roadmaps for automated guided vehicle systems. *IEEE International Conference on Industrial Engineering and Engineering Management*, *2016-December*, 977–981. https://doi.org/10.1109/IEEM.2016.7798023

Uttendorf, S., & Overmeyer, L. (2015). Fuzzy-enhanced path-finding algorithm for agv roadmaps. *2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology (IFSA-EUSFLAT-15)*, 675–681.

Van Nuffelen, C. (1976). On the incidence matrix of a graph. *IEEE Transactions on Circuits and Systems*, *23*(9), 572–572.

Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, *170*, 677–709. https://doi.org/10.1016/j.ejor.2004.09.020

Watanabe, M., Furukawa, M., & Kakazu, Y. (2001). Intelligent agv driving toward an autonomous decentralized manufacturing system [10th International Conference on Flexible Automation and Intelligent]. *Robotics and Computer-Integrated Manufacturing*, *17*(1), 57–64. https://doi.org/https://doi.org/10.1016/S0736-5845(00)00037-5

Williams, H. P. (2009). Integer programming. In *Logic and integer programming* (pp. 25–70). Springer.

Zhang, Z., Guo, Q., Chen, J., & Yuan, P. (2018). Collision-free route planning for multiple agvs in an automated warehouse based on collision classification. *IEEE Access*, *6*, 26022–26035.

Appendix A

A.I Introduction

An example of a virtual AGV in movement is illustrated in Figure A.1. The suitcase laying on the AGV can leave the vehicle in both vertical directions. Figure A.2 shows an example of an AGV system including the paths over which they drive.

Fig. A.1: Virtual AGV

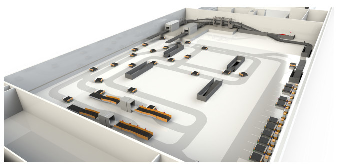


Fig. A.2: Example AGV system

A.I.1 Performance AGV System Based on Pathmap: Consider an environment in which example AGV 1 moves packages A between the green locations and AGV 2 moves packages B between the red locations, as visualised in Figure A.3. Several different path maps can be designed to deliver the correct packages at the correct locations by the correct AGVs. In Figure A.3, a path map with an intersection for which the direction of both paths is upwards is shown. If the AGVs are designed such that the flow for both AGV 1 and AGV 2 can merge perfectly, none of the AGVs needs to wait for the other. Note that this path map is not efficient since all vertices are visited is an AGV finishes a delivery task. A different path map is visualised in Figure A.4. Now, the direction of the crossing contains opposite directions resulting in different throughput performances. In Figure A.5, again, a path map that can successfully distribute the requested flows is shown, but without an intersection. Those example graphs show that all design choices can individually influence the overall performance of the system.

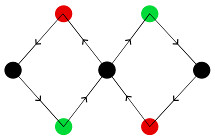


Fig. A.3: Map 1

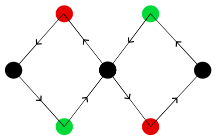


Fig. A.4: Map 2

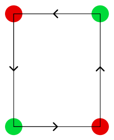


Fig. A.5: Map 3

The previously discussed examples show differences in design choices for intersections. Many more design choices besides path intersections exist. For example, the influence of hindering paths on the overall throughput. The two opposite directed loops visualised in Figure A.6 could result in a problem, when looking at the vehicle shape colored in blue. The questions of how and if the AGVs can even find a way out of this situation arise, can the AGVs handle reverse driving, or is this situation an example of a deadlock? In the second scenario, shown in Figure A.7, the direction of both loops is the same. The design of this path map, therefore, results in a completely different performance. Other examples that could influence the performance of an AGV system are: the merging and diverging of paths.

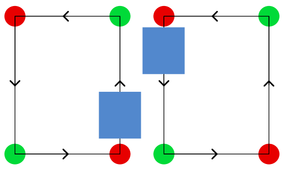


Fig. A.6: Loop scenario 1

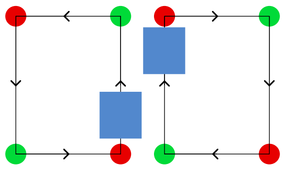


Fig. A.7: Loop scenario 2

A.I.2 Proposed Methodology: Figure A.8 shows a visualisation of the proposed methodology per stage. The left and right upper corners refer to the practical problem converted into a mathematical model. Accordingly, the mathematical solution is obtained after validation (right lower corner). Lastly, this is converted into the practical solution in the left lower corner.

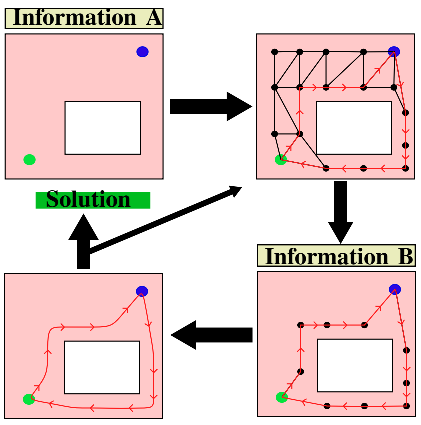


Fig. A.8: Proposed methodology

Appendix B

B.I Algorithmic Framework

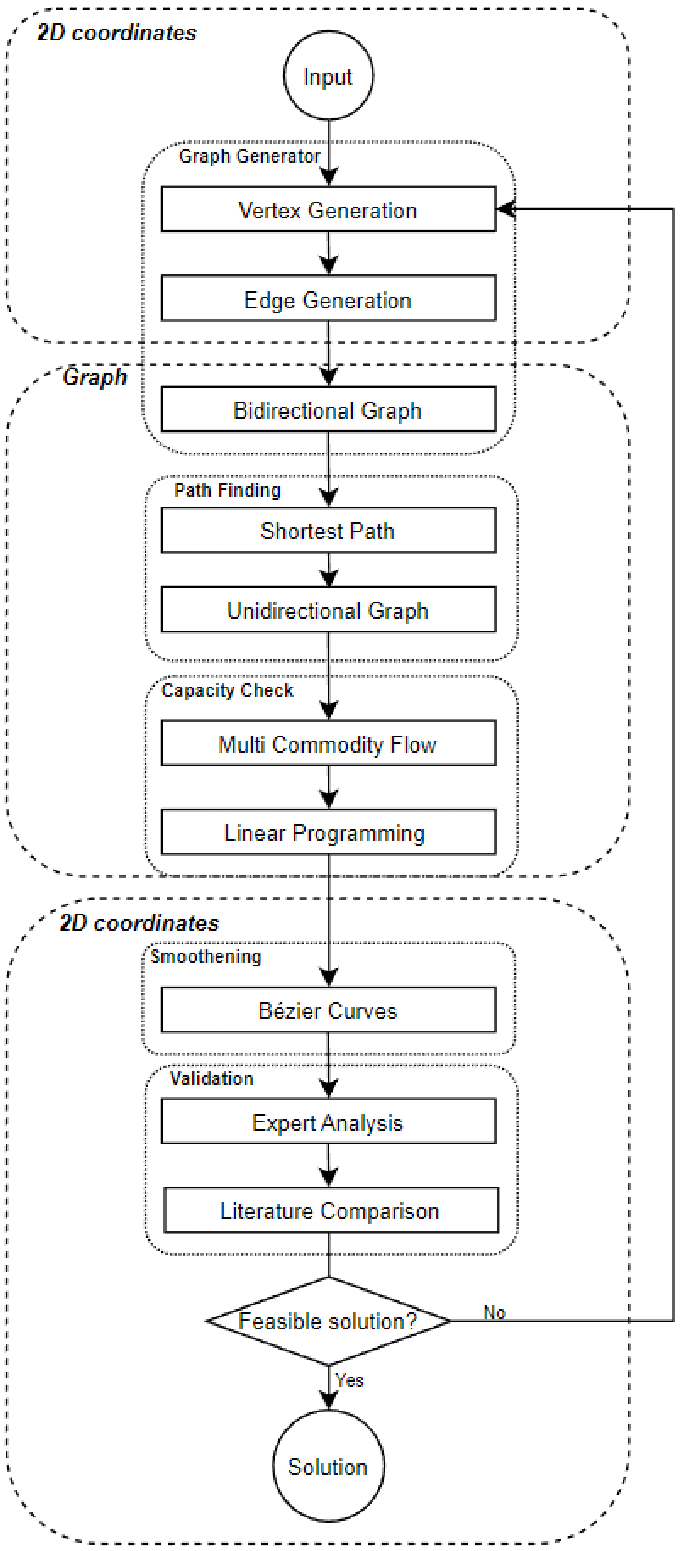


Fig. B.1: Algorithmic framework

B.II Input Layout

In Figure B.2, an example of a fictional layout is shown. The grey surface represents the drivable space with a safety margin, or clearance distance, visualised in pink. AGVs can drive in the safety margin, but no paths can be drawn here. Obstacles (non-drivable space) are depicted as white surfaces. The purple and blue dots refer to pickup and drop-off points. Note that they can be reached via two angles each, as the arrows corresponding to the headings illustrate.

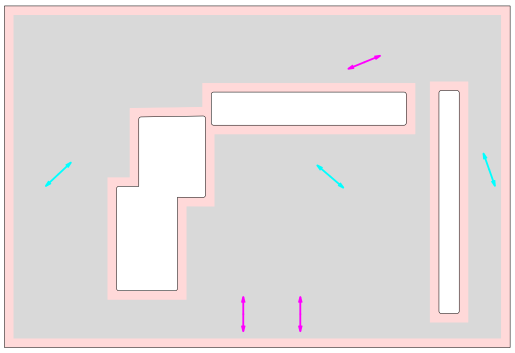


Fig. B.2: Example fiction input layout

B.III Vertex Generation

Figure B.3 illustrates an example input layout with equally spaced and divided generated vertices, denoted by the small blue dots. Note the additional orientation handling points generated in the extension of the heading for each handling point, this is explained later in this study. The same layout, but with vertices generated randomly, is shown in Figure B.4.

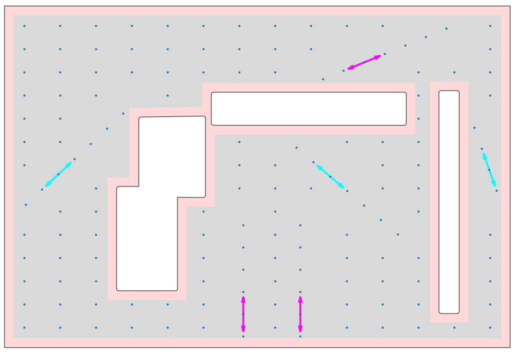


Fig. B.3: Layout with evenly divided and equally spaced vertices

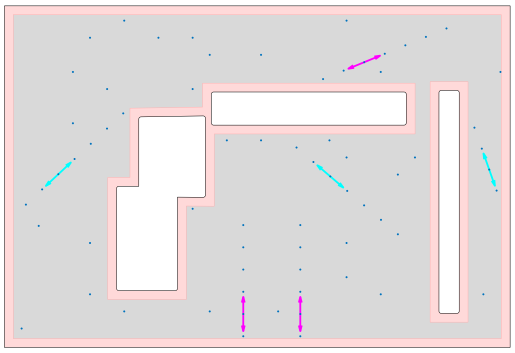


Fig. B.4: Layout with randomly placed vertices

B.III.1 Orientation Handling Point Vertices: An example of a drop-off point (cyan coloured) with its two orientation handling points, visualised in dark blue.

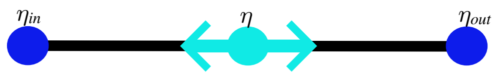


Fig. B.5: Handling point with its orientation handling points

Consider the drivable space (note that safety margins are not visualised in this example) in the upper image of Figure B.6. A handling point with its corresponding orientation handling point vertices is visualised in cyan and dark blue respectively. Additionally, general vertices are visualised by the black dots and two obstacles are illustrated in white. In this situation, the handling point orientation nodes can not be reached because of the distances between the grid-like generated vertices. As visualised by the grey dotted lines, all possible connections to other nodes intersect with an obstacle.

In situations like this, generating more orientation handling point vertices makes the graph feasible by creating the possibility to connect general vertices with the rest of the graph.

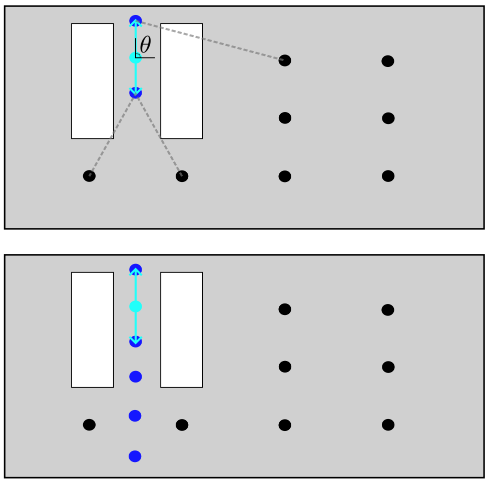


Fig. B.6: Illustration of additional vertices aligned with θ

B.IV Edge Generation

In this section, two examples of edge generation are discussed: Delaunay triangulation and Voronoi diagrams.

B.IV.1 Delaunay Triangulation: Figure B.7 visualises Delaunay triangulation in blue through vertices in red. The circles intersecting with the vertices are colored gray.

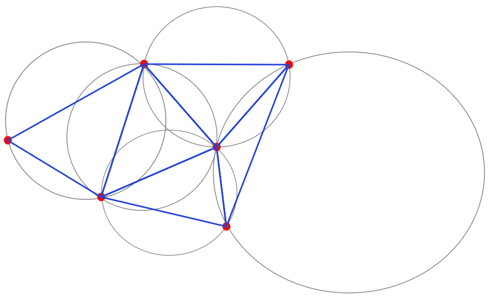


Fig. B.7: Delaunay triangulation visualised in blue

B.IV.2 Voronoi Diagram: A Voronoi diagram divides, given a plane and a set of vertices V, according to the nearest-neighbour (or closest region principle), a surface into several so called 'Voronoi regions' (Aurenhammer, 1991). The formal definition of the Voronoi diagram is shown in Equation 13, for which R_k, P_k, and P_j represent the Voronoi cell or region, and the sites (points in space) associated with indices k and j respectively. Furthermore, $d(a, b)$ refers to the Euclidean distance between a and b.

$$R_k = \{v \in V \mid d(v, P_k) \leq d(x, P_j)\ \forall j \neq k\} \tag{13}$$

Figure B.8 illustrates an example of a set of edges generated with Delaunay triangulation in blue in combination with the Voronoi diagram in green, given a set of vertices presented in red. Note that the connection of the centers of all circumcircles results in the Voronoi diagram.

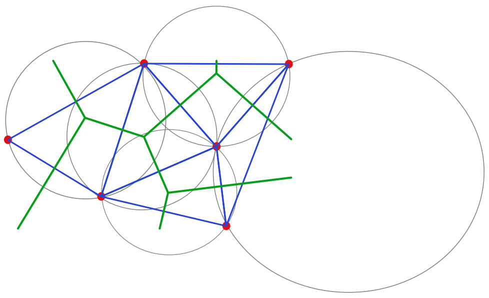


Fig. B.8: Voronoi diagram visualized in green

A disadvantage of Voronoi diagrams is that the polygon of the drivable space in which the Voronoi diagram should hold needs to be restricted. An example of edges in an unbounded environment is presented in Figure B.9. One method of tackling this issue is by initializing new vertices around the boundary edges of the drivable space, as discussed by Sainlot et al., 2017. When using Delaunay triangulation, this additional implementation is not needed.

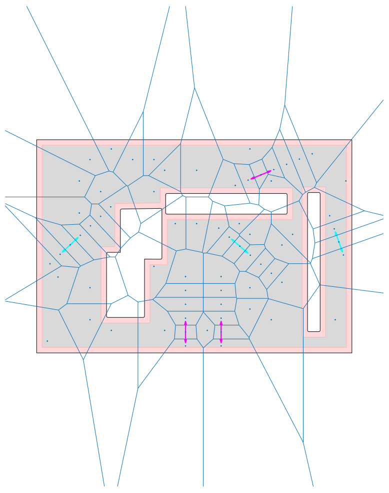


Fig. B.9: Example layout with Voronoi diagram

B.IV.3 Non-Unique Delaunay Triangulation: Consider a situation in which four unique vertices lie on the same circle in a 2D plane. When following the Delaunay triangulation principle, one finds two solutions that fulfill the criteria. Figure B.10 illustrates this example of Delaunay triangulation being non-unique, with vertices and edges represented in red and blue respectively.

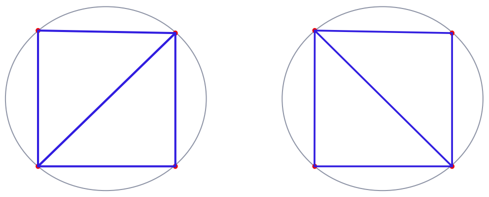


Fig. B.10: Example of non-unique Delaunay triangulation

With the Delaunay triangulation not being unique, one could say that the algorithm itself is non-deterministic, which means that with the same input different solutions could be obtained. Therefore, it is decided to model the triangulation in situations similar to Figure B.10 always as in the right situation, the diagonal is drawn from the upper left to the lower right corner.

B.IV.4 Undirected or Bidirectional Graph: An undirected or bidirectional graph $\overline{G}$ is shown in Figure B.11.

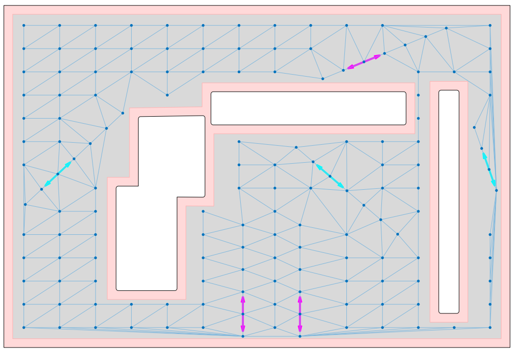


Fig. B.11: Undirected graph in input layout

B.V Mathematical Solutions

B.V.1 Loop and Bypass Generator: The methods *Loop* and *Bypass* generator are described in Algorithm 1 and Algorithm 2.

Algorithm 1 Loop Generator
```
Input:    Undirected graph $\overline{G}$
Output:   Directed graph $G$
Notation: Handling point: $\eta$, orientation handling point one and
          two: $\eta_{\text{in}}$, $\eta_{\text{out}}$, vertices and edges: $V$ and $E$
1: for All $\eta$ do
2:     Generate directed path from $\eta_{\text{in}}$ to $\eta_{\text{out}}$ through $\eta$;
3: end for
4: for All $\eta_{\text{out}}$ do
5:     Connect to nearest $\eta_{\text{in}}$ to which $\eta_{\text{out}}$ is not connected yet;
6:     Delete used $\eta_{\text{in}}$ as option for other $\eta_{\text{out}}$'s;
7:     Store all visited $V$ and $E$ as $G = (V, E)$;
8: end for
```

Algorithm 2 Bypass Generator
```
Input:    Undirected graph $\overline{G}$
Output:   Directed graph $G$
Notation: Handling point: $\eta$, orientation handling point one and
          two: $\eta_{\text{in}}$, $\eta_{\text{out}}$, vertices and edges: $V$ and $E$, pickup
          and drop-off points: $\mathcal{P}$ and $\mathcal{D}$, number of bypasses: $n$
1: for All $\eta$ do
2:     Generate directed path from $\eta_{\text{in}}$ to $\eta_{\text{out}}$ through $\eta$;
3: end for
4: for All $\eta_{\text{out}}$ do
5:     Connect to nearest $\eta_{\text{in}}$ to which $\eta_{\text{out}}$ is not connected yet;
6:     Delete used $\eta_{\text{in}}$ as option for other $\eta_{\text{out}}$'s;
7:     Store all visited $V$ and $E$ as $G = (V, E)$;
8: end for
9: if $\eta \in \mathcal{P}$ then
10:    for All $n$ $\eta_{\text{in}}$'s do
11:        Connect to random $\eta_{\text{out}}$ for $\eta \in \mathcal{D}$;
12:        Delete used $\eta_{\text{out}}$ as option for other $\eta_{\text{in}}$'s ;
13:        Store all visited $V$ and $E$ in $G$;
14:    end for
15: else if $\eta \in \mathcal{D}$ then
16:    for All $n$ $\eta_{\text{out}}$'s do
17:        Connect to random $\eta_{\text{in}}$ for $\eta \in \mathcal{P}$;
18:        Delete used$\eta_{\text{in}}$ as option for other $\eta_{\text{out}}$'s ;
19:        Store all visited $V$ and $E$ in $G$;
20:    end for
21: end if
```

A visualisation for both the loop and bypass generator is shown in Figure B.12. The four numbered vertices represent handling points, note that for this example orientation handling points are not included. Furthermore, the black lines represent an undirected graph. The directed blue path segments represent the generated loop via Algorithm 1. In addition, the directed paths generated via Algorithm 2 are visualised in green and they connect vertices two and three to vertices one and four such that several options to move through the directed graph are possible.

In this example, there is no distinction between pickup and drop-off points. In the algorithm, this is taken into account such that no unnecessary paths from pickup point p_i to p_j (loaded vehicle to a pickup point) or drop-off point d_k to d_l (empty vehicle to drop-off point) are generated. This is also the case for additional bypasses.

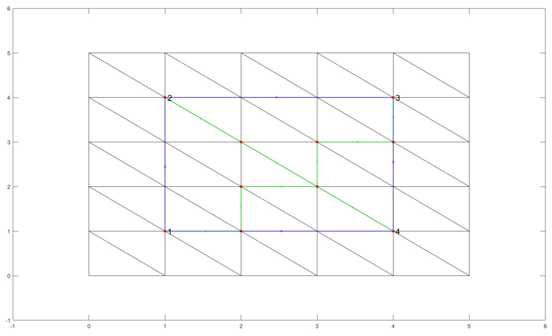


Fig. B.12: Generated loop (blue) and bypass paths (green) visualised

B.V.2 Multiple Loops Generator: In this subsection, the algorithm for generating k-loops based on k-nearest neighbours is described in Algorithm 3, followed by visualisations for different values of k.

Algorithm 3 k-Nearest Handling Point Loops Generator
```
Input:    Undirected graph $\overline{G}$
Output:   Directed graph $G$
Notation: Handling point: $\eta$, orientation handling point one and
          two: $\eta_{\text{in}}$, $\eta_{\text{out}}$, vertices and edges: $V$ and $E$, number
          of neighbours to connect $\eta$ with: $k$ , pickup and
          drop-off points: $\mathcal{P}$ and $\mathcal{D}$
1: for All $\eta$ do
2:     Generate directed path from $\eta_{\text{in}}$ to $\eta_{\text{out}}$ through $\eta$;
3: end for
4: if $\eta \in \mathcal{P}$ then
5:     Find $k$ nearest $\eta \in \mathcal{D}$;
6:     for All $\eta_{\text{out}} \in \mathcal{P}$ do
7:         Connect to $\eta_{\text{in}}$ of $k$ nearest $\eta \in \mathcal{D}$
8:         Store all visited $V$ and $E$ in $G$;
9:     end for
10: else if $\eta \in \mathcal{D}$ then
11:    Find $k$ nearest $\eta \in \mathcal{P}$;
12:    for All $\eta_{\text{in}} \in \mathcal{D}$ do
13:        Connect to $\eta_{\text{out}}$ of $k$ nearest $\eta \in \mathcal{P}$
14:        Store all visited $V$ and $E$ in $G$;
15:    end for
16: end if
```

A possible solution for $k = 1$ that is not strongly connected is shown in Figure B.13. It is not possible to traverse between the two generated loops. Again, for this example, no distinction is made between pickups and drop-offs. The real model does distinguish both types of handling points. Two illustrations for k equal to one and two are shown in Figure B.13 and Figure B.14.

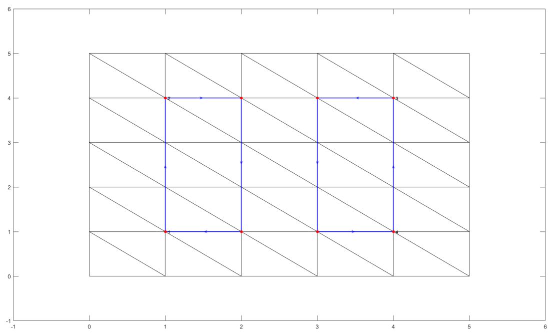


Fig. B.13: Possible solution for k = 1

Figure B.14 illustrates a possible solution for the four handling points with k equal to two. It is shown that this adjustment already makes the directed graph strongly connected.

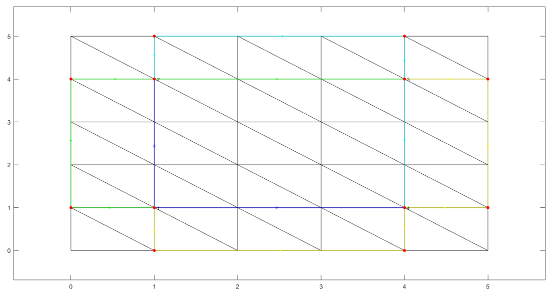


Fig. B.14: Possible solution for k = 2

In the previously discussed examples, edges are unidirectional. The algorithm uses edge lengths as initial weights (edge e_{uv} has weight w_{uv}). Additionally, paths through handling points are initialized to respect their headings. Once a path is used, its opposite directed path (starting graph is bidirectional), is removed from the directed graph in order to avoid bidirectional paths.

B.VI Dijkstra's Algorithm

Dijkstra's algorithm is a graph search method that finds optimal (shortest) paths given a graph $G = (V, E)$ (Dijkstra et al., 1959). All edge weights must be strictly positive, otherwise, it is recommended to use the Bellman-Ford algorithm (more time-consuming, see Ari Muzakir, 2020, for an explanation). In Dijkstra's method, priority queue Q is sorted according to a cost-to-come function $C : X \to [0, \infty]$. The costs per vertex visited are computed by taking the sum of all costs to reach the corresponding vertex. For example, for vertex v, expanded from vertex u, the cost-to-come function for vertex v equals $C(v) = C^*(u) + w(u, v)$, in which $C^*(u)$ refers to the optimal cost to reach vertex u (LaValle, 2006).

An example of Dijkstra's algorithm is provided with Figure B.15. This graph consists of six vertices r, s, t, u, v, and w and seven directed edges with their corresponding weights denoted by the numbers in the figure.

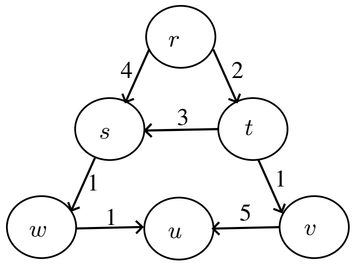


Fig. B.15: Example directed graph

Algorithm 4 shows the solution to reach vertex u starting at vertex r given the directed graph in Figure B.15 including notation and steps. The bold vertices are removed as cheaper paths to the same vertex exist already.

Algorithm 4 Example Dijkstra applied on graph

```
Notation: Expanded — Q
1: Empty      — $r(PreviouslyVisitedVertex, 0)$
2: $r$          — $t(r, 2)s(r, 4)$
3: $rt$         — $v(t, 3)s(r, 4)\mathbf{s(t, 5)}$
4: $rtv$        — $s(r, 4)u(v, 8)$
5: $rtvs$       — $w(s, 5)u(v, 8)$
6: $rtvsw$      — $u(w, 6)\mathbf{u(v, 8)}$
7: Optimal solution: $path(r \to s \to w \to u)$ with costs 6
```

B.VI.1 Features: The ability to generate many path layouts is included in the framework through the features stated below:

- Orientation of headings (plus or minus π rad)
- Number of generated vertices
- Type of vertex generation: randomly placed or grid-like
- Number of additional orientation handling point vertices
- Number of generated bypasses
- Number of k-nearest neighbours
- Avoiding high occupancy rate per edge by minimizing or maximizing the number of paths

As discussed earlier, the heading of each handling point is acceptable from two directions, θ and θ rotated by π. Varying the headings of all handling points results in different solutions. Accordingly, the method for vertex generation is an important factor in what the graph looks like. Both the number of vertices and the way of generating them (randomly or grid-based), are therefore variables for generating different solutions.

Furthermore, the variables for the number of additional orientation handling point vertices generated bypasses, and nearest neighbours also result in different solutions. The last implemented feature focuses on edge occupancy and path minimization. Is it important that edges are used efficiently (high occupancy rate) or should the efficiency be kept low? This variable introduces a trade-off between highly occupied paths versus more diverges, crossings, and merges.

B.VII Linear Programming Return Flow

Consider graph $G = (V, E)$, as presented in Figure B.16 in which the green vertex α represents a pick-up spot, blue vertices β and δ refer to drop-off points, and the red vertex γ stands for a general vertex. Additionally, the directed edges connecting the vertices are visualised in black. The following two commodities are requested:

1) Flow from vertex α to vertex β should be at least k_1
2) Flow from vertex α to vertex δ should be at least k_2

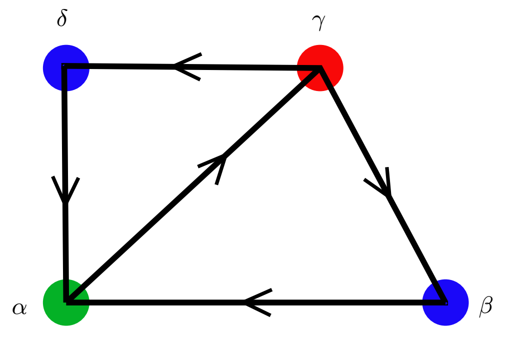


Fig. B.16: Directed graph

Since two commodities are requested in this example, the graph can be split into two separate graphs laying on top of each other, one for each commodity, as shown in Figure B.17. This does not change anything for the coordinate system since both networks are graphs (not geometrical maps), it is only split for visualisation of the different commodities in a graph.

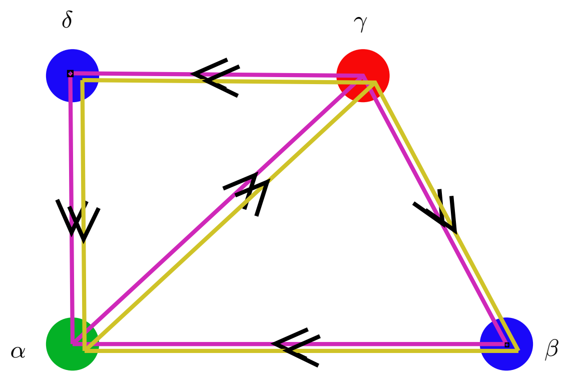


Fig. B.17: Graph split into two imaginary graphs, one per commodity

With the use of Equation 3, the flow leaving pick-up point α for commodity k_1 is described as minus the demand for the corresponding commodity: $-k_1$. Now consider the second commodity k_2, as flow leaving the same pick-up point α but for a different commodity is then equal to $-k_2$. The total flow leaving pick-up point α equals $-k_1 - k_2$. Regarding the flow at the drop-off points, first, commodity k_1 is looked into. When using Equation 4 for drop-off point β, the flow equals $+k_1$. The same holds for commodity k_2 and drop-off point δ, the flow at this vertex equals $+k_2$.

Figure B.18 illustrates the directed graph with the flow per commodity. The blue and green line segments refer to commodities k_1 and k_2, respectively. Note that also the flow conservation constraint, as described in Equation 2, holds for vertex γ, all flow incoming goes out.

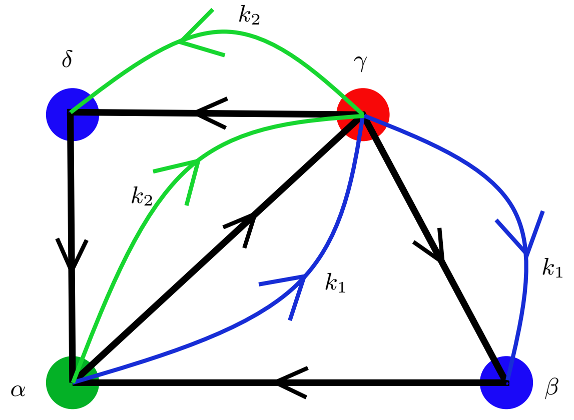


Fig. B.18: Directed graph with flow per commodity

In Figure B.18, it is shown that the return flow constraint is not included yet, the flow stops moving at drop-off points. This means that flow should be generated at pick-up point α if more flow from pickup point α is requested. One way of solving this problem is to generate the 'empty-AGV' or auxiliary return flow, let us call this commodity return commodity k_r. The visualisation for the additional return flow is presented in Figure B.19 by the red line segments.

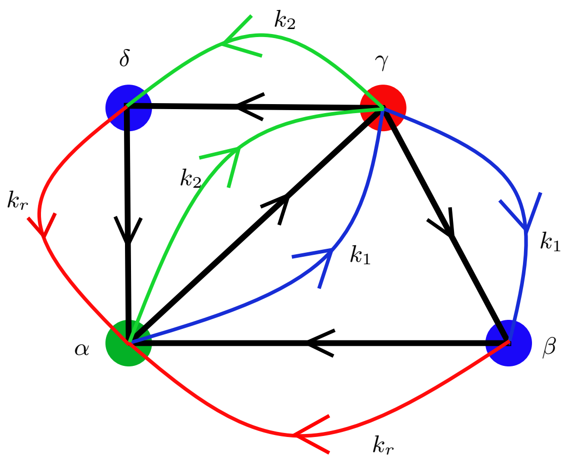


Fig. B.19: Directed graph with return commodity k_r

Now that the entire loop is closed, the flow conservation holds for each vertex and so for the entire directed graph. The return commodity k_r for pickup vertex α is thus described by: $-(-k_1 + -k_2) = k_1 + k_2$. As discussed earlier, outgoing flows are defined as negative, and incoming commodities as positive. The negative sign before adding the negative (outgoing) commodities compensates for the flow conservation for vertex α.

In equation form, the return flow commodity can be computed with Equation 14. For each drop-off vertex $d \in D$, the return commodity is determined by the reversed difference between incoming ($k_{i,d}$) and outgoing commodities ($k_{o,v}$).

$$\forall d \in D: \qquad k_{r,d} = -(k_{i,d} + k_{o,v}) \tag{14}$$

One final remark related to the feasibility of solutions including the return flow. Let us consider the maximum edge capacity for each edge in the graph to be identical. Then, this example only holds when $c(e) \geq k_1 + k_2$, if this is not the case, then the first constraint is not satisfied for edge $e(\alpha, \gamma)$, as explained in Equation 1.

B.VIII Objective Function Linear Programming

The graph illustrated in Figure B.20 is an extension of Figure B.16, two vertices ϵ and ζ connected by four edges (2, 3, 8, and 9) are added to provide an example of the influence on how coefficient vector f is chosen in solving the linear programming problem.

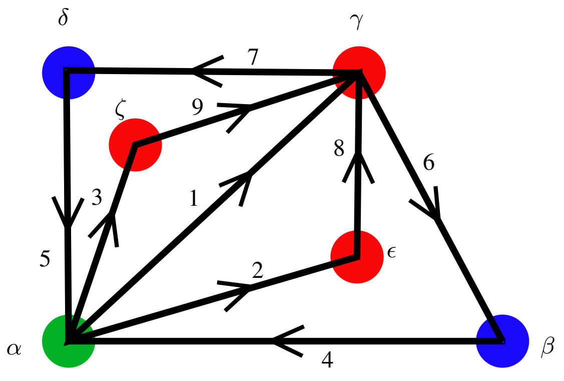


Fig. B.20: Directed graph

Considering the graph illustrated in Figure B.20, all three options for coefficient vector, as discussed in section II.C, are explained. Figure B.21, Figure B.22, and Figure B.23 contain tables with the flow per edge $e \in E$ per commodity $k \in K$ on the left, and the visualised flow per commodity in the graph on the right. The colors in the graphs are defined as follows: black for unused edges, green and yellow for commodities k_1 and k_2, and blue edges for the auxiliary return flow commodity k_r. Before going into detail on those tables and figures, the input for the three models will be discussed.

To solve the linear programming problem, first, a graph is defined. Second, a flow matrix, $\mathbb{F} = (f_{ij}) \in \mathbb{R}^{n \times n}$ with n equal to the number of handling points in the graph, is created. The requested input flow equals k_1 and k_2 moving from pick-up α to drop-off vertices β and δ with both k_1 and k_2 equal to one, as shown in Equation 15. Furthermore, the maximum edge capacity is set equal to two. The rows and columns represent all handling points. Thus, f_{ij} refers to the flow target from handling point i to handling point j.

$$\mathbb{F} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{15}$$

The results for f equal to a vector filled with ones are shown in Figure B.21. As discussed before, this setting finds the minimum flow using the fewest edges possible. Both the green and yellow flow edges show that the shortest paths are found towards the drop-off vertices starting from pick-up vertex α. This is followed by the return flow back to vertex α. The additional paths from α to γ consisting of more edges are not used to solve the problem and thus has the shortest path been found.

	$\mathbf{k_1}$	$\mathbf{k_2}$	$\mathbf{k_r}$
Edge 1	1	1	0
Edge 2	0	0	0
Edge 3	0	0	0
Edge 4	0	0	1
Edge 5	0	0	1
Edge 6	1	0	0
Edge 7	0	1	0
Edge 8	0	0	0
Edge 9	0	0	0

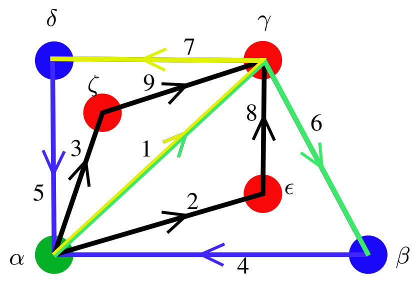


Fig. B.21: Table and flow visualisation for f equal to ones

In the solution for the second option, the maximum flow per edge is found, as shown in Figure B.22. Instead of using edge 1 only, now additional edges 2, 3, 8, and 9 are used to move the flow from α to γ. Furthermore, the maximum edge capacity is reached. Since for the requested commodities k_1 and k_2 equal to one, the edge capacity is not entirely used, now additional flow is added to the system moving through the entire directed graph. All edges that are in use now exactly match the maximum edge capacity of two.

	$\mathbf{k_1}$	$\mathbf{k_2}$	$\mathbf{k_r}$
Edge 1	0	0	0
Edge 2	2	0	0
Edge 3	0	2	0
Edge 4	1	0	1
Edge 5	0	1	1
Edge 6	2	0	0
Edge 7	0	2	0
Edge 8	2	0	0
Edge 9	0	2	0

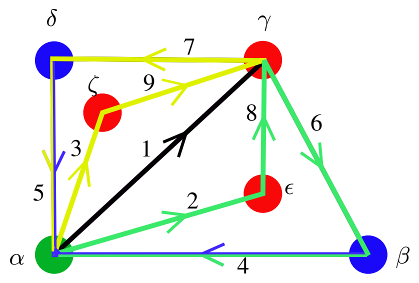


Fig. B.22: Table and flow visualisation for f equal to *min* ones

The third option results are presented in Figure B.23. As illustrated in the graph, a solution that neither uses the minimum or maximum edge capacity is found. This chosen f-vector only validates if a feasible solution exists, it does not matter what solution.

	$\mathbf{k_1}$	$\mathbf{k_2}$	$\mathbf{k_r}$
Edge 1	0	0	0
Edge 2	0	0	0
Edge 3	1	1	0
Edge 4	0	0	1
Edge 5	0	0	1
Edge 6	1	0	0
Edge 7	0	1	0
Edge 8	0	0	0
Edge 9	1	1	0

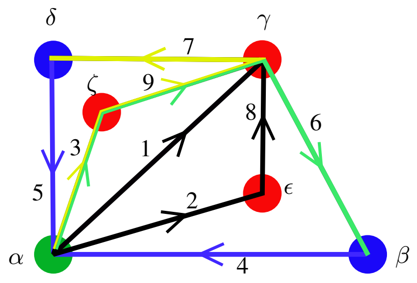


Fig. B.23: Table and flow visualisation for f equal to zeros

The objective function results are summed up below. Considering the goal of the solver to minimize objective function $f^T x_{e,k}$, the results show that f equal to minus ones finds best solutions.

- f equal to ones: $f^T x_{e,k} = 6$
- f equal to minus ones: $f^T x_{e,k} = -16$
- f equal to zeros: $f^T x_{e,k} = 0$

Those results are expected because of the definitions of the coefficient vector f. Furthermore, f equal to ones tries to find the minimum flow and edges are used so therefore $|f_{+1}^T x_{e,k}| < |f_{-1}^T x_{e,k}|$.

B.IX Solver Algorithm

The difference between the regular simplex method and the dual simplex method is that the first method searches a given space of basic (non-optimal) solutions in a greedy way toward either infeasibility, unboundedness, or an optimal solution (Koberstein, 2005). The dual simplex method starts with an optimal but infeasible solution and works towards feasibility (Huerta and de Lara, 2015). Since in this study, feasibility is more important than optimality, the dual simplex method is preferred.

The interior-point algorithm generates an initial point and iteratively approaches the optimal solution from the interior of the feasible set of solutions. This means that the worst-case complexity is larger compared to the dual simplex method, that only considers the boundaries of the feasible set (Robere, 2012). Therefore, the proposed algorithm makes use of the dual simplex method.

The dual simplex and interior-point are tested on three different input graphs: G_1, G_2, and G_3, in which the number of vertices and edges increase from graph one to graph three. Both methods are analysed by the metrics computation time and objective function results, as presented in Table B.1 and Table B.2. The differences in computation time are negligible and the costs are slightly less for the dual simplex method for all three inputs. This complements the point made by Andersen and Andersen, 2000, the interior-point method returning slightly less accurate solutions within a slightly shorter computation time. Differences could be the result of noise or rounding differences within the iterative computations for the interior-point method. However, this study does not contain further investigation in this field of interest.

	Dual Simplex [s]	Interior-point [s]
G_1	0.012	0.012
G_2	0.021	0.021
G_3	0.026	0.024

TABLE B.1. Computation time per method and input

	Dual Simplex $f^T x_{e,k}$	Interior-point $f^T x_{e,k}$
G_1	-16	-16
G_2	-2355	-2324
G_3	-2955	-2875

TABLE B.2. Objective function per method and input

B.X Bézier Curves

Figure B.24 shows a Bézier curve that smoothens the corner denoted by corner points P_1, P_2, and P_3. The grey dotted lines represent edges from a graph and the black line shows the Bézier curve. Note that the curve from this example is symmetric, that does not necessarily have to be the case when generating Bézier curves.

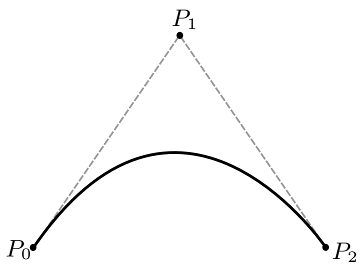


Fig. B.24: Symmetric Bezier Curve

B.XI Path Intersections

An example of intersection modeling in a graph and the geometrical view is shown in Figure B.25. In the left image, the crossing above the handling point visualised in cyan contains four edges. From a graph perspective, this counts as one crossing. In the geometrical path map image, visualised on the right, the number of intersections of path segments above the handling point equals two merges and one crossing: three intersections in total. In this study, intersections are counted in the graph structure, as presented in the left image.

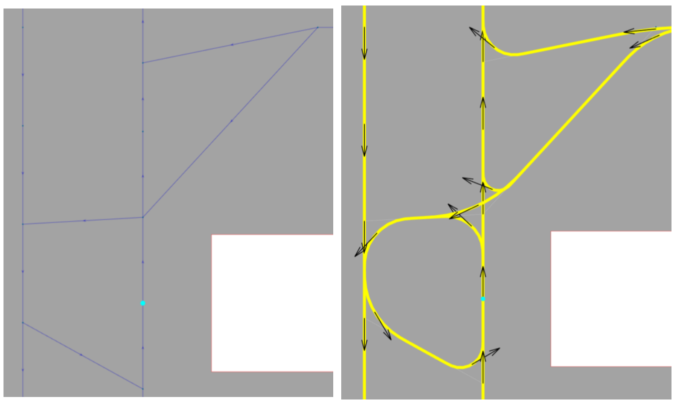


Fig. B.25: Path intersection modelling

B.XII Control Points Determination

Given corner points P_{start}, P_{trans}, and P_{end}, the unit tangent vectors t_s and t_e along the linear edges are defined as Equation 16

$$t_s = \frac{P_{\text{trans}} - P_{\text{start}}}{|P_{\text{trans}} - P_{\text{start}}|} \quad \text{and} \quad t_e = \frac{P_{\text{end}} - P_{\text{trans}}}{|P_{\text{end}} - P_{\text{trans}}|} \tag{16}$$

All six control points can be expressed in corner points, distances c and d, and unit tangent vectors t_s and t_e, as illustrated in Equation 17. For further derivation, consult Sencer and Shamoto, 2014.

$$\begin{aligned} P_0 &= P_{\text{trans}} - (2c+d)t_s & P_5 &= P_{\text{trans}} + (2c+d)t_e \\ P_1 &= P_0 + ct_s & P_4 &= P_5 - ct_e \\ P_2 &= P_0 + 2ct_s & P_3 &= P5 - 2ct_e \end{aligned} \tag{17}$$

B.XIII Symmetric Bézier Curves

Consider the corner determined by corner points P_{start}, P_{trans}, and P_{end}, expressed by the linear grey and dotted lines. After applying Bézier curves for smoothing the corner it does not result in a symmetric curve such that G^2 continuity is not achieved.

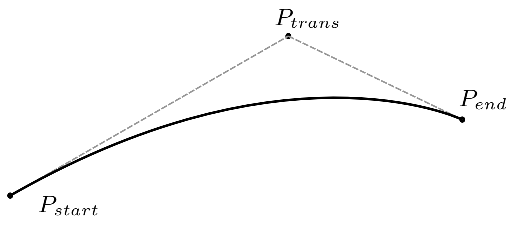


Fig. B.26: Asymmetric Bézier Curve

One method for creating a symmetric Bézier curve is to split up the edges shaping the corner. Let the length of edge $e(P_{\text{start}}, P_{\text{trans}})$ and $e(P_{\text{trans}}, P_{\text{end}})$ be defined as $l_{s,t}$ and $l_{t,e}$. Now divide the longest edge of $l_{s,t}$ and $l_{t,e}$, which is $l_{s,t}$ in this example, into two line segments l_1 and l_2 with $l_1 = l_{s,e}$ such that two symmetric Bézier curves can be generated, an example is shown in Figure B.27.

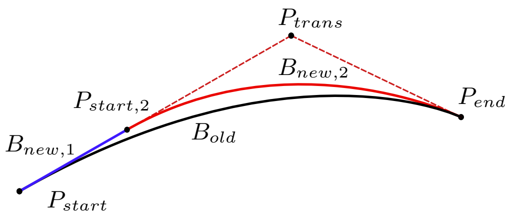


Fig. B.27: Symmetric Bézier Curves

A new cornering point ,$P_{\text{start},2}$, is added such that the old curve B_{old} is split into two new curves: $B_{\text{new},1}$ and $B_{\text{new},2}$. The second Bézier curve is a 5^{th} order curve with corner points $P_{\text{start},2}$, P_{trans}, and P_{end} in which edges $e(P_{\text{start},2}, P_{\text{trans}})$ and $e(P_{\text{trans}}, P_{\text{end}})$ are identical resulting in $B_{\text{new},2}$ being symmetric (visualised by the red line in Figure B.27). $B_{\text{new},1}$ is called the connecting Bézier curve, shown by the blue line segment in Figure B.27. This curve is a first-order Bézier curve, which is a straight line (Sampaio, 2017). This method must be applied to the longest edge spanning the corner.

B.XIV Corner Points Determination

Observe graph G with vertices $v_1, \ldots, v_5$ and edges $e_1, \ldots, e_4$ visualised in black in Figure 7 as an example of a directed graph. If the corners need to be smaller and the curve should lay for at least half the length of the edge on top of the edge itself, first new vertices are generated that can be used as corner points. In Figure B.28, new vertices on each $e_i \in E$ are generated exactly at the middle of each edge, denoted by $V_1, \ldots, V_4$. The yellow line segments show the connection between V_i and V_j for all newly generated vertices.

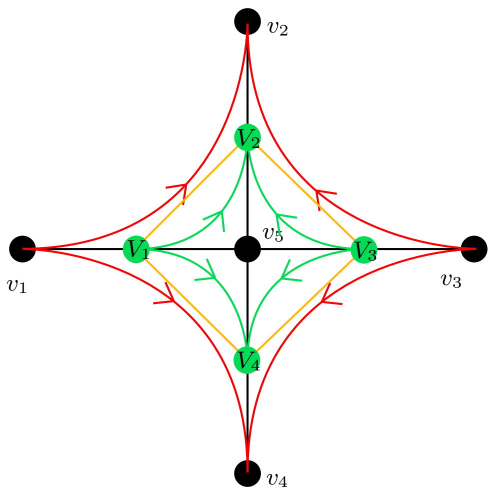


Fig. B.28: Visualisation new vertices and curves

Once the new vertices have been generated, again, the curve will be split up into three parts, the straight line from v_1 to V_1, the curve from V_1 to V_2, and the straight line from V_2 to v_2. Those three segments replace the old curve starting at v_1 and ending at v_2.

Another problem that is solved by applying this method of smoothing, is the problem of consecutive corners. Two consecutive corners lead to discontinuity in the curves, as illustrated in Figure B.29. By adding vertices V_1, V_2, and V_3 at the middle of all edges and defining new corner points before smoothing the curves, the discontinuity is avoided as shown in Figure B.30.

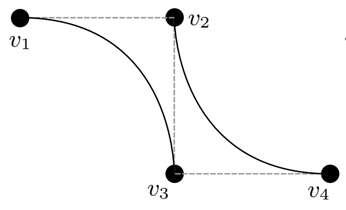


Fig. B.29: Discontinuous path

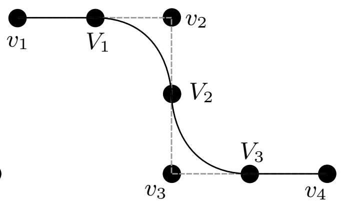


Fig. B.30: Continuous path

B.XV Optimization Curvature

Sencer and Shamoto, 2014, discuss a power regression to estimate the curvature optimal n value for a wider range of cornering angles θ. Here, n refers to the ratio between lengths c and d in control point determination: $n = c/d$. Consequently, the optimal ratio is computed by Equation 18.

$$n(\theta) = \frac{1}{2.0769}\theta^{0.9927} \tag{18}$$

Since cornering angle θ is a constant for each corner, only the ratio between lengths c and d is known. If one of the lengths is fixed, still one of them remains to be specified. In the proposed algorithm, it is chosen to specify d and compute c with the use of Equation 18. Then, d is calculated such that control points P_0 and P_5 remain on the line segments spanning the corner. This is followed by choosing d such that $d + 2c$ equals the minimum of distances P_{start} to P_{trans} and P_{trans} to P_{end} (Teusink, 2022).

Appendix C

C.I Fictional Instances

C.I.1 Airport Layout: Figure C.1 illustrates the airport instance with drivable space colored in gray, five pickup points colored in magenta, and four drop-off points colored in cyan. The obstacles, represented in white, are non-drivable spaces. A safety boundary of half the AGV size surrounds each obstacle such that no paths too close to obstacles or outlines are generated.

The obstacle in the center of the layout represents a pair of screening lines through which luggage should be sent from pickup to drop-off points. Another screening line is located above this pair of lines. The squared-shaped obstacles illustrate the start (for drop-offs) and end (for pickups) points of conveyor belts over which the baggage should traverse after and before the screening. Only a small amount of baggage is required to be sent through the system per hour such that the requested capacity is relatively low.

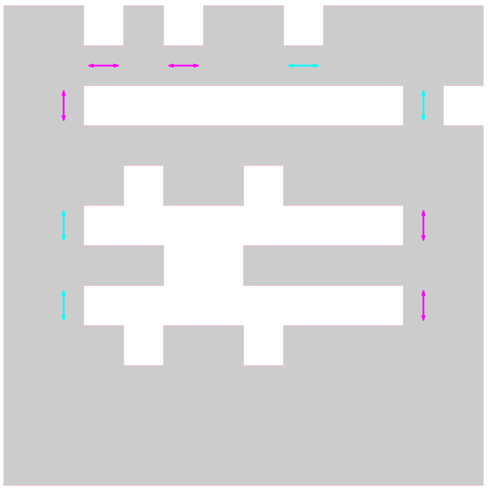


Fig. C.1: Aiport instance

C.I.2 Parcel Grid Layout: The parcel grid shown in Figure C.2 contains 8 pickups and 16 drop-offs. All square-shaped obstacles represent a place in which packages are dropped off or picked up.

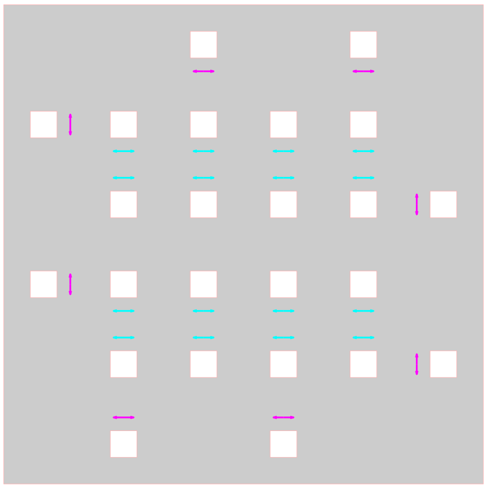


Fig. C.2: Parcel grid instance

C.I.3 Generation Setup: The generation setup for the fictional instances is shown in Table C.1. The flow targets are defined as a flow from each pickup vertex to all drop-off vertices. Furthermore, AGVs are modeled as squared vehicles with length L. This length is used for safety boundaries around non-drivable space and for vertex generation. The maximum edge capacity is expressed by $\mathbf{e_{max}}$. The cost function weights based on the goals of the AGV system per instance, as explained in section III.A.1 and section III.A.2.

Instance	Flow targets	L	$\mathbf{e_{max}}$	Weights
Airport	1 $[units/hour]$	1.3 $[m]$	100 $[units/hour]$	λ_1 = 10 λ_2 = .5 λ_3 = .25 λ_4 = 2.5
Parcel	15 $[units/hour]$	1.3 $[m]$	600 $[units/hour]$	λ_1 = 5 λ_2 = 1 λ_3 = 5 λ_4 = .25

TABLE C.1. Generation setup for fictional instances

For the airport instance, the weight factor for the path utilization is four times the weight factor for the redundancy such that the importance of both components is almost in the same order of values. The targets within the parcel instance are focused on a high occupancy rate within a small footprint of the path in the drivable space. Consequently, λ_1 and λ_3 are the highest values of all weights.

C.II Comparing Algorithm

The algorithm discussed in Teusink, 2022, from now on called the exact algorithm, solves the path map design problem as a discrete optimization problem with an exact algorithm. It is concluded that for small instances optimal solutions, and for large instances feasible solutions are found. The main goal of the exact algorithm, optimality, differs from this study, feasibility. In the exact algorithm, no fractional flow can be sent through, edges are either used completely (1) with the required flow targets or not at all (0). Additionally, a difference has been made between the maximum edge capacity for a straight line and a corner segment. Since the overall flow of a system is determined by the edge with the least maximum edge capacity, this assumption does not match reality. A straight-line segment can hardly reach this capacity if it is constrained by 'bottlenecks' with less maximum edge capacity. Furthermore, in the graph generation stage, no diagonal edges between vertices are allowed, only horizontal and vertical edges. This drastically decreases the number of possible paths. Consequently, this could result in a limitation of feasibility in some situations. A more practically usable implementation of the exact algorithm is the sweep of AGVs over paths. This property validates are hindering path curves influencing flow on other segments resulting in more detailed and realistic flow distribution computations.

C.III Generation Results Airport Instance

C.III.1 Parameter Explanation: Several different generation setups are tested. First of all, the number of k-neighbors for the multiple loop generator and the number of bypasses n are varied from one up to four (the number of pickup points in the layout). Then, the number of initially generated vertices is tested for two different cases: 250 and 300 vertices. In Figure C.3, the difference between those implementations is shown for the corresponding instance. The left configuration (250 vertices) generates vertices in between the obstacles resulting in a possible shortcut.

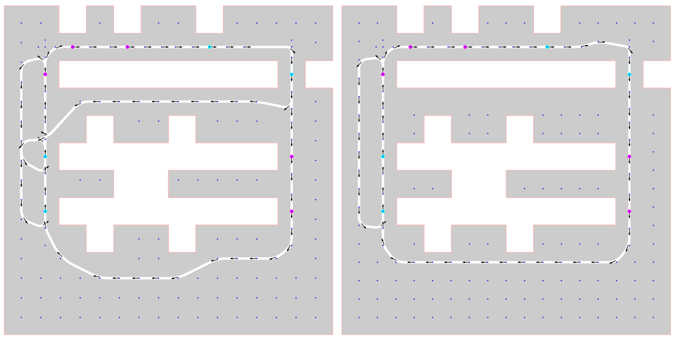


Fig. C.3: 250 vertices (left) versus 300 vertices (right)

Another parameter resulting in quite different solutions is minimizing the number of paths, this results in fewer but higher occupied path segments. If this parameter is not important, all highly occupied paths are pulled apart such that a larger percentage of the layout is filled with more but less occupied paths. An example is illustrated in Figure C.4, all other parameters are identical. For the left layout, it is set to be important to have more paths with lower occupancy and for the right layout it is not.

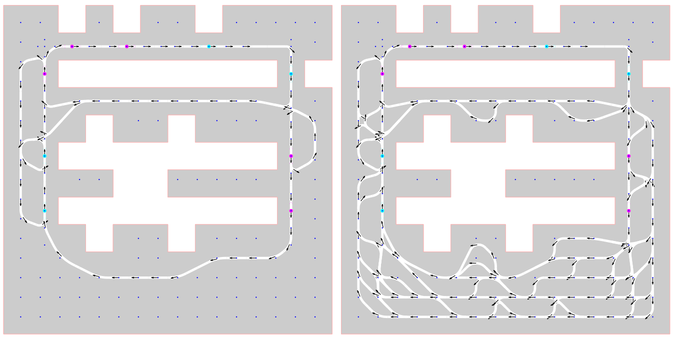


Fig. C.4: Use of many paths (left) versus few paths (right)

Note that for all generations in this section the orientation of headings, as mentioned in section B.VI.1, is kept constant and defined such that a clockwise loop could be drawn through all handling points.

C.III.2 Loop Generator Results: The first results are all generated by the loop generator. For all generations, this results in a clockwise loop. Note that for the loop generator no intersections are generated, therefore, the parameter for many paths versus few paths does not change anything for the solutions. For this instance, the weights for the cost of space utilization and redundancy are the highest. Table C.2 shows the costs for all four generations. It can be concluded that the solutions for 300 vertices have fewer costs. More vertices result in a denser grid such that the path can surround the obstacles more closely which results in less path length used.

	250 vertices	300 vertices
Grid	4.0566	**3.9983**
Random	5.0773	4.9826

TABLE C.2. Cost for loop algorithms

Figure C.5 shows the results for both types of vertex generation for 250 vertices. In Figure C.6, the results for 300 vertices can be observed. Comparing both equally spaced vertices solutions, using 300 vertices results in paths more strictly around the obstacles and a lower path length. The same holds for the randomly placed vertex solutions.

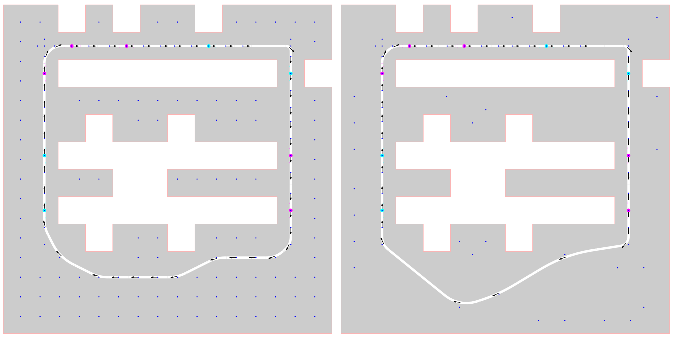


Fig. C.5: Loop results for 250 vertices

One remarkable path segment for the equally spaced, 300 vertices, is the small kink in the graph at the right top corner. Due to the vertex generation, apparently, a vertex in the extension of the heading of all handling points on that line has been removed due to safety distance constraints. The solution with the least costs is visualised by the left layout in Figure C.5. The redundancy costs are equal for all four solutions such that path utilization determines the total cost for those layouts.

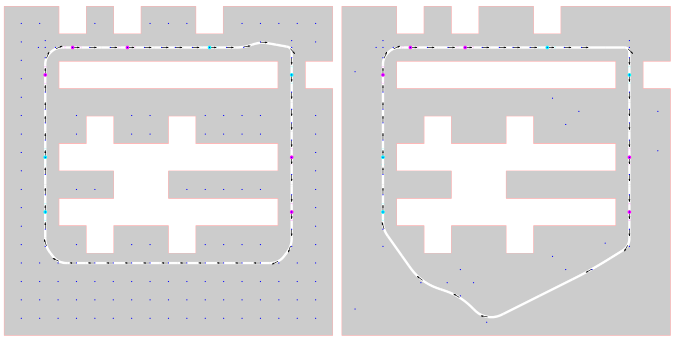


Fig. C.6: Loop results for 300 vertices

C.III.3 Paths Minimization Configuration Results: In this subsection, the results for the bypass and multiple loop generators with varying k, n, and the number of vertices are discussed. All configurations are forcing the algorithm to generate the least number of path segments. Table C.3 summarizes the total cost for all generations. The first thing that points out, is that grid vertices have fewer costs if a feasible solution exists. A possible reason for this could be the possibility of adding a path in between the two central obstacles, which is beneficial for the redundancy of the layout. When using randomly placed vertices, no path in between the obstacles can be drawn in some configurations. Another possible reason could be the choice of the number of vertices, it could be that 250 vertices are not ideal for this instance. Lastly, again, the density of the grid and the (random) distance to obstacles could influence the performance.

k	Grid	Random
1	∞	5.4397
2	**4.0023**	5.6422
3	**4.0023**	5.6422
4	**4.0023**	5.6422

n	Grid	Random
1	4.0566	5.0773
2	4.0566	5.0773
3	4.262	5.5316
4	4.142	5.5316

TABLE C.3. Cost for 250 vertices and avoid additional paths true

One remarkable finding is that $k = 1$ for the grid-spaced vertices has infinite costs (the flow distribution is not possible, thus the solution is infeasible according to the validation by the linear programming problem). Figure C.7 shows the reason for this, the path is not strongly connected since the input required the handling points to only connect to one closest (other type) handling point. Because of the path through the obstacles, the entire path is not connected such that the handling point located at the right side of the layout at middle height, can never be reached. The same input, but with randomly generated vertices instead, is visualised in Figure C.8. This solution is feasible because no cheaper path through the obstacles exists. Therefore, it is cheaper to connect the loops around the obstacles such that all handling points can be visited.

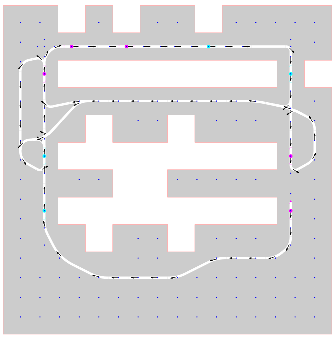


Fig. C.7: Infinite cost

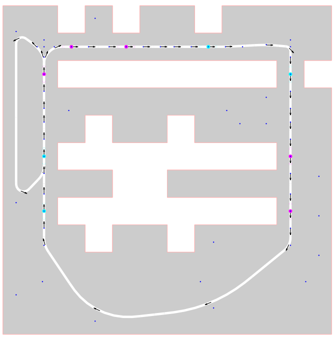


Fig. C.8: Connected

In Figure C.9 and Figure C.10, the solutions for k equal to two and n equal to four are shown, for which the first mentioned has the least total costs. It can be observed that both solutions make use of a path in between the central obstacles such that alternative paths between handling points arise. Furthermore, Figure C.9 uses slightly more space but has an additional path from one of the two pickups located on the right side of the layout. Apparently, this improvement in terms of redundancy weighs more than the additional few paths that are being used as expressed in the total costs.

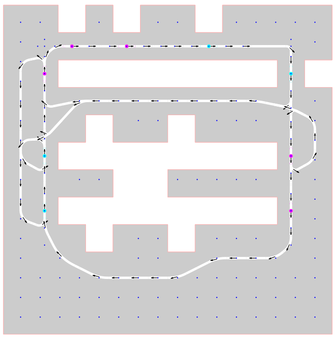


Fig. C.9: $k = 2$

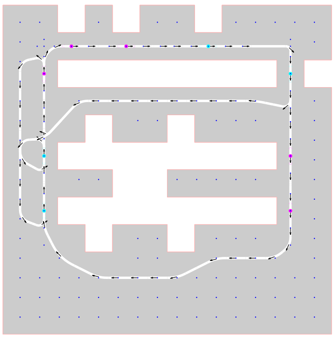


Fig. C.10: $n = 4$

Figure C.11 and Figure C.12 illustrate the two solutions with the largest costs. For both layouts bypasses or cycles are added that do not improve the redundancy enough, considering the additional space they utilize. Both additional cycles improve the accessibility leaving from one handling point only (left top corner). For all the other handling points it does not add anything in terms of redundancy, resulting in the path utilization costs taking over.

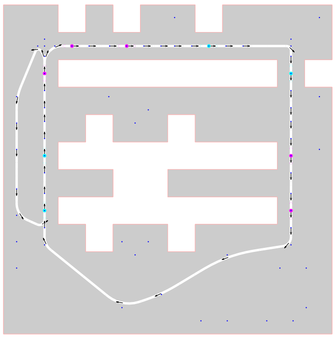


Fig. C.11: $n = 4$

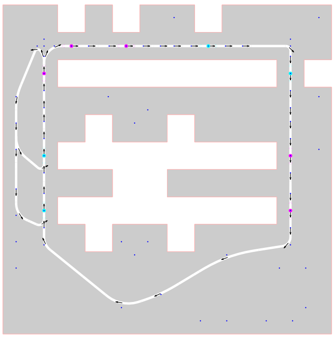


Fig. C.12: $k = 4$

One last remarkable result that is obtained from Table C.3 is that for several solutions the total costs are the same. The reason for this is that because it is preferred to generate the least number of paths with high occupancy rates, space utilization is important. Therefore, often increasing either k or n results in exactly the same solution because the paths to connect handling points already exist. If new handling point connections are required after increasing n or k, existing paths are reused. The reason why most solutions remain feasible is due to the relatively low (1 unit per hour) flow targets for this instance in combination with a high maximum edge capacity (100 units per hour). So in terms of capacity, a path map for this instance is always feasible if it is strongly connected.

Increase the number of initial vertices up to 300: Now the results for the same generation setup but with more initially generated vertices (300) are presented. The total costs for all generations are shown in Table C.4.

k	**Grid**	**Random**
1	4.1801	∞
2	4.2432	6.1942
3	4.2432	6.1942
4	4.2432	6.1942

n	**Grid**	**Random**
1	**3.9983**	5.5058
2	**3.9983**	5.5058
3	4.1972	6.0897
4	4.1972	6.0897

TABLE C.4. Cost for 300 vertices and avoid additional paths true

A noticeable aspect here, is that for 300 vertices and k equal to one, now the grid-placed vertices turned feasible and the randomly placed vertices became infeasible instead, as shown in Figure C.13. The reason for this is that the vertices generated in between the obstacles for the grid-placed vertices do not exist anymore, as explained earlier. Meanwhile, for the randomly placed vertices, they do exist now.

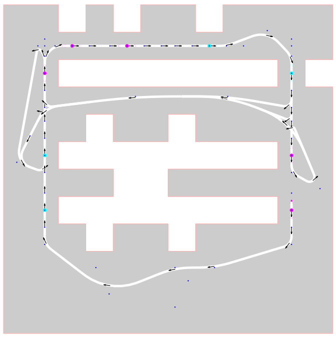


Fig. C.13: Not connected

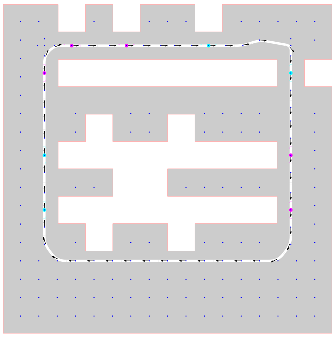


Fig. C.14: Least cost

Figure C.14 visualises the solution with the least costs for this generation. In all solutions, the additional redundancy costs do not compensate for the additional path utilization they require. Therefore, the best-performing solution in terms of costs is a basic loop in which no additional crossing paths are used. The second and third best-performing layouts are visualised in Figure C.15 and Figure C.16, using the k-loops generator for the left, and the bypass generator for the right solution. As shown, the additional cycle added to the left side of the loop around the obstacles makes the difference. Figure C.15 adds another bypassing path such that two additional drop-off points can be reached from the pickup point just in front of the diverging path. The loop shown in Figure C.16 does not have a shortcut to both drop-off points. Both paths are performing better in terms of redundancy compared to the simple loop, but also more path segments are used.

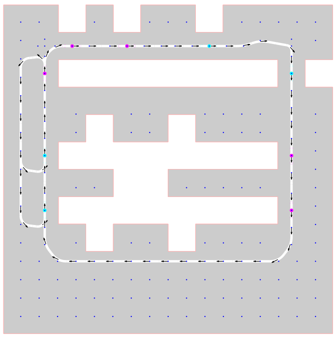


Fig. C.15: $k = 1$

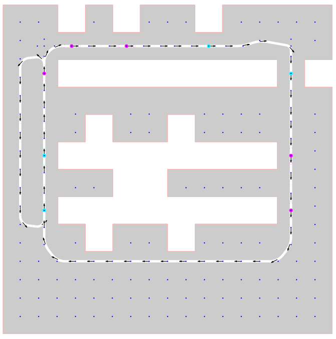


Fig. C.16: $n = 3$

C.III.4 No Minimization of Paths Configuration Results: For the solutions in this section, the framework is forced to generate more paths in drivable space such that intersections have an overall lower occupancy. Besides that, all of the same scenarios from the previously discussed section are tested. In Table C.5, the results are presented for 250 vertices. No infeasible solutions are generated in this configuration. Configurations with random vertex generation result in higher costs.

k	Grid	Random
1	4.9485	7.0155
2	6.0743	7.4473
3	6.5078	7.7323
4	6.7202	7.8631

n	Grid	Random
1	**4.0566**	5.2722
2	4.5066	5.5584
3	4.6094	6.3043
4	4.8425	6.9652

TABLE C.5. Cost for 250 vertices and avoid additional paths false

In Figure C.17 and Figure C.18, the solutions with the second least (the best performing solution is the loop again) and most costs are visualised. The high costs for Figure C.18 are the result of the amount of space that is used. The narrow corridor at the top of the layout has enough space for one path only. Therefore, the profit that could be made from the redundancy is limited, which in the end results in too large costs overall.

In Figure C.17, a better balance between space and redundancy is created. With fewer paths, the redundancy has improved a lot. The additional lane from the right bottom corner to the left upper corner looks promising, however, without any additional bypasses, it limits its own power. The performance increases because it creates the possibility to overtake the two cyan handling points on the left. But, when one or more bypasses are added just in front of one of the three vertically aligned left vertices the performance would be much better considering redundancy and the additional space needed.

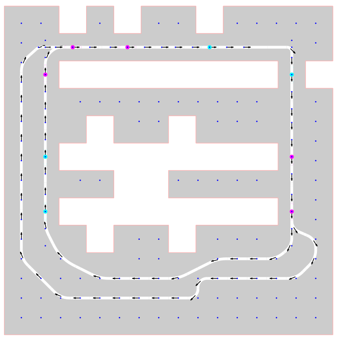


Fig. C.17: $n = 2$

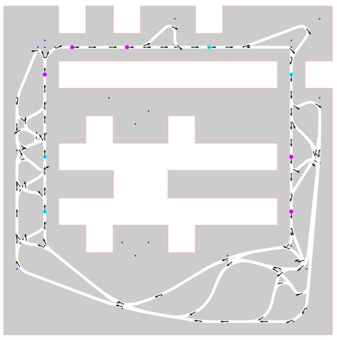


Fig. C.18: $k = 4$

A solution that provides preferable shortcuts such that the overall layout is more redundant is shown in Figure C.19. It is clearly visible that much more new paths are generated to visit and surpass specific handling points. The downside of this layout is the drastically enlarged footprint. A layout containing a bypass through the two obstacles in the middle is shown in Figure C.20. Besides this, it has several unnecessary additional paths, starting with the two small loops attached to the path through these obstacles.

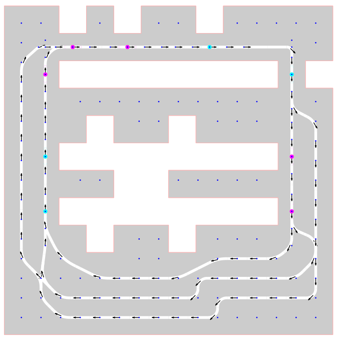


Fig. C.19: $n = 3$

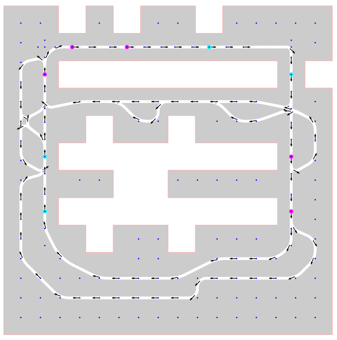


Fig. C.20: $k = 1$

Increase the number of initial vertices up to 300: The generation results for 300 vertices are shown in Table C.6. Again, the performance for the layouts based on randomly generated vertices is worse. Also, the overall performance of the bypass generator is better compared to k-loops.The least costs are found with two bypasses in a grid-like graph.

k	Grid	Random
1	4.5755	7.2976
2	5.1811	7.3653
3	5.4695	7.4634
4	5.7835	7.4634

n	Grid	Random
1	3.9983	5.5058
2	**3.4895**	5.8619
3	4.0021	6.7506
4	4.5293	6.9635

TABLE C.6. Cost for 300 vertices and avoid additional paths false

The least cost solution is illustrated in Figure C.21. Comparing this solution to Figure C.17, it can be seen that the additional loop is slightly tighter, which makes the path utilization for this part preferable. The additional loop at the right top corner shows the difference between the cost function result and the use of common sense. According to the cost function, this is beneficial for redundancy in terms of costs. However, an engineer would probably define this loop as useless due to its small path length shortly before and after handling points. Figure C.22 shows the most cost for randomly placed vertices in general, with $k = 4$. Again, the multiple-loop method generates too many unnecessary paths.

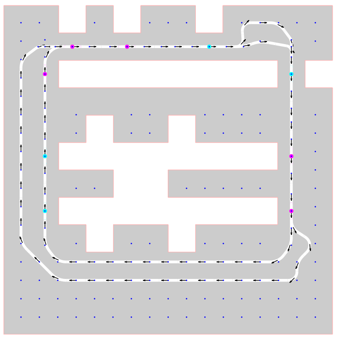


Fig. C.21: $n = 3$

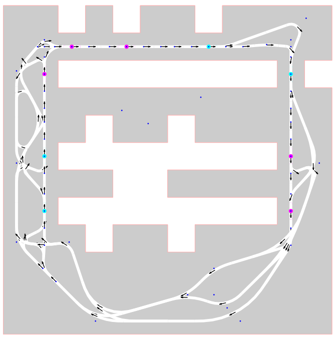


Fig. C.22: $k = 4$

C.III.5 Overview of Best and Worst Solutions Airport Instance: The scores and computation times for the three best and worst performing layouts are shown in Table C.7 and Table C.8.

Generation Setup	Score	Time [s]
Aim for min. paths Bypass, $n = 2$ 300 vertices (grid)	3.4895	1.547
Aim for min. paths Loop 300 vertices (grid)	3.9983	1.551
Aim for min. paths k-loops, $k = 2$ 250 vertices (grid)	4.0023	1.409

TABLE C.7. Three best performing solutions airport instance

Generation Setup	Score	Time [s]
Do not aim for min. paths k-loops, $k = 4$ 250 vertices (random)	7.8631	1.029
Do not aim for min. paths k-loops, $k = 3$ 250 vertices (random)	7.7323	1.005
Aim for min. paths k-loops, $k = 4$ 300 vertices(random)	7.4634	1.015

TABLE C.8. Three worst performing solutions airport instance

C.III.6 Exact Algorithm Layouts Airport Instance: Figure C.23, Figure C.24, Figure C.25, and Figure C.26 show the results generated by the exact algorithm for comparison. When analyzing the results, they somehow look similar but it is clearly visible that only necessary paths are included in the path map due to the (0,1) integer solution.

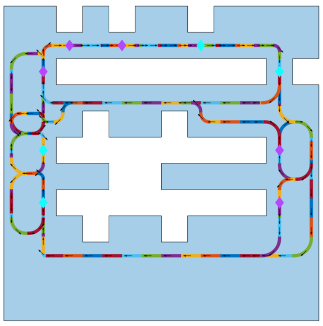


Fig. C.23: Solution 1

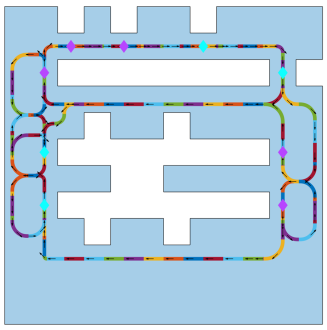


Fig. C.24: Solution 2

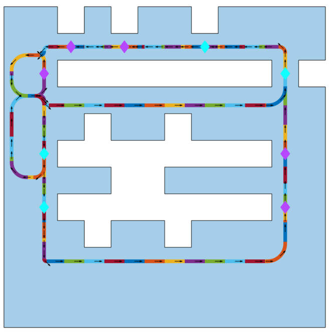


Fig. C.25: Solution 3

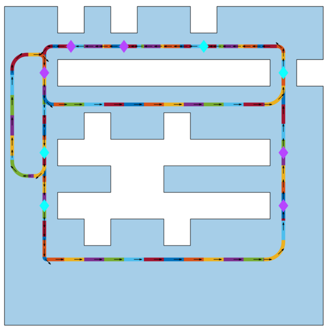


Fig. C.26: Solution 4

The total costs for all four layouts are computed and summarized in Table C.9. The first solution performs best according to the cost function with the corresponding weight factors.

Solution	Cost
1	2.7870
2	3.0645
3	3.4356
4	3.3895

TABLE C.9. Score for comparison layouts

C.III.7 Comparison Frameworks Airport Instance: The cost distributions for each component of the best-performing solutions for both frameworks are shown in Table C.10.

	This study	Exact algorithm
Total cost	3.4895	2.7870
Path utilization cost	$\lambda_1$0.2254	$\lambda_1$0.1617
Path intersection cost	$\lambda_2$0.0230	$\lambda_2$0.0645
Occupancy cost	$\lambda_3$0.8979	$\lambda_3$0.8000
Redundancy cost	$\lambda_4$0.4000	$\lambda_4$0.375
Computation time	1.551 [s]	212.3925 [s]

TABLE C.10. Comparison between cost components

The results show that for this instance and setup, the proposed framework performs worse than the exact algorithm. The only component for which the exact algorithm performs worse is path intersection. Considering that both layouts can handle the requested flow targets given the generation settings, the computation plays an important role. Each solution generated by the exact algorithm has a computation of around 150 times larger. Both path maps are visualised in Figure C.27 and Figure C.28.

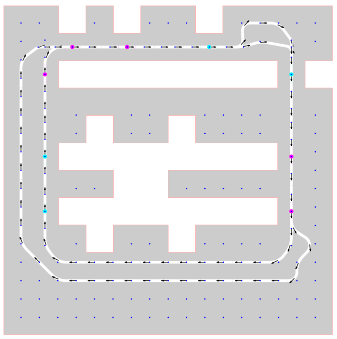


Fig. C.27: This study

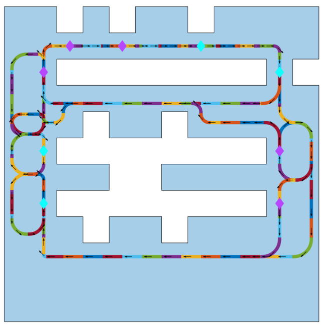


Fig. C.28: Exact algorithm

A possible reason for the difference in path utilization is graph generation. One aspect that differs in graph generation is how vertices can be connected. The proposed algorithm makes use of diagonal connections and the exact algorithm does not. Obviously, those additional paths increase the total possible path length. In layouts containing small corridors, diagonal paths might be unnecessary, as not many options for connection exist. This results in larger costs according to the definition of the path utilization cost component in this study. The total possible path length for Figure C.27 equals 532 meters compared to the 841 meters in Figure C.28. The used path lengths differ as well. The proposed algorithm uses 141 meters compared to the 136 meters of the exact algorithm.

The difference in total possible path lengths can be observed from Figure C.29 and Figure C.30. The graph used for the exact algorithm is much denser and the safety boundary around obstacles and close to the outline of the drivable space is not yet taken into account in this phase of computation. Due to real-world implementations, such as hindering paths, additional vertices are needed to change paths such that hindering can be avoided. Therefore, because of the different types of graphs, the path utilization or footprint costs are not entirely fair to compare.

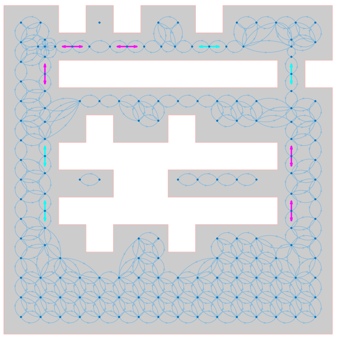


Fig. C.29: This study

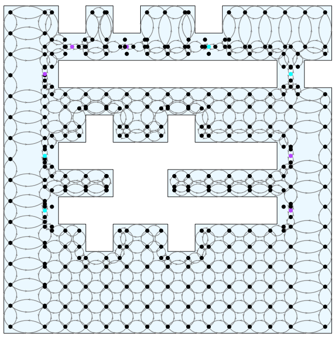


Fig. C.30: Exact algorithm

As graph generation is an important step in path map generation, it is chosen to use the total possible path length over the drivable space for footprint costs. Generating the graph for the exact algorithm already takes 28.4975 seconds, which is around twenty times the total computation time of the proposed framework. Within the process of graph generation, the trade-off between computation complexity and time, and the inclusion of additional features exists. The exact algorithm chose to use a more intelligent but computationally expensive generation.

Considering the path intersections, as Figure C.27 and Figure C.23 show, the left solution contains two merges only, compared to 8 merges and intersections in the right layout, resulting in larger costs. Furthermore, the occupancy costs for the exact algorithm are less since fewer paths are used for the same flow targets, meaning that this path map is more efficient. Lastly, it is obvious that more alternative paths from and toward handling points exist in the solution for the exact algorithm, making this solution more redundant.

C.IV Changing Other Parameters

In this section, an example of the influence of changing other parameters than previously discussed in the configurations is illustrated. In all previously explained generations, both k and n are varied from one up to four, to connect all pickup points at least to k and n drop-off points. This parameter can also be used the other way around, starting with connecting drop-offs to pickups first. In this example, one forces the path through the obstacles, as visualised in Figure C.31.

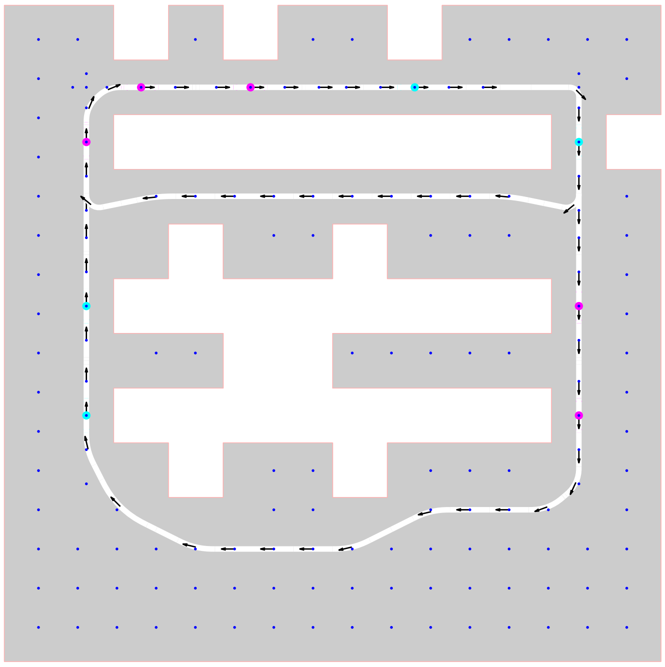


Fig. C.31: Forcing path through obstacles, $n = 1$

After analysing the previous scores and results, it turns out that forcing the path through the obstacles could drastically improve the performance. When changing the order of connecting handling points from drop-off to pickup points, the result shown in Figure C.31 is generated. The power of this framework is that many parameters can easily be changed and combined such that one can tune the framework towards an expected solution.

C.V Experts Analysis

An analysis of the best-performing solution concerning the cost function (Figure C.27) is made from the point of view of experts in the field of path map design. The first remark is on the small additional loop at the right top corner. Small additional loops can be extremely beneficial, for example, in switching lanes to avoid a traffic jam. However, they drastically influence the overall throughput because additional merges arise. It would rather be beneficial to add a loop between the two screening lines. Furthermore, merging just in front of a handling point is killing for throughput, this should be avoided. The form of simplicity given the requirements is a strength of this generated layout. In terms of redundancy, this layout does not perform well. When adding at least three bypasses in the double lane, one for each handling point, the redundancy would be improved a lot. Also, additional smoothing would be beneficial to exclude some irregular path segments where smooth segments seem more logical. This improved the overall throughput since AGVs do not need to brake and accelerate when this is not needed.

C.VI Generation Results Parcel Instance

C.VI.1 Parameter Explanation: The generation setups for the parcel instance differ slightly from the setups used for the airport instance. Within the airport instance setups, the heading for all handling points is set constant. For this instance, different heading setups will be used to test the influence. The number of initially generated vehicles for those generations is a constant number of 500 vertices. The same other variable parameters are used in generation and analysis.

C.VI.2 Loop Generator Results: With the additional option for varying the headings of the handling points, different solutions for the loop generator arise. Considering the target flows and the maximum edge capacity, it is expected that the loop generator will not result in feasible solutions. The motivation for this can be explained using Equation 19, for which a simplification of the flow leaving a handling point is computed. Given flow target $t_i(p, d_i)$ for pickup point p to drop-off point d_i equal to 15 items per hour, and the number of drop-off points, D, equal to 18, the flow over edge e right after the pickup point can be computed and is equal to 270 units per hour.

$$c_p(e) = \sum_{i=1}^{D} t_i(p_i, d_i) \tag{19}$$

This means that when the flow leaves three pickups the edge limit of 600 units an hour is exceeded. This instance contains eight pickup points, and generating a loop without enough options for the flow to diverge, will not work. The results while using the loop generator are shown in Table C.11. All solutions, but one, are infeasible.

	Fewest paths		*Most paths*	
	Grid	**Random**	**Grid**	**Random**
Non-ordered	∞	∞	**4.5551**	∞
Ordered	∞	∞	∞	∞

TABLE C.11. Costs for all loop generated solutions

The only solution in which enough bypasses are included is with a non-ordered set of handling points, vertices generated as a grid while minimizing the used paths. The result is shown in Figure C.32. In terms of theoretical feasibility, this solution performs well. However, when looking at the layout more closely, it is not expected to perform well in reality. For example, almost none of the drop-off points are connected to each other such that most of them can only be reached while using individual and long paths just meant for the corresponding destination. For an AGV, complicated and different paths for different drop-off points should be used with many curves and intersections. Furthermore, this solution uses almost all available space, this is also not matching the requirements.

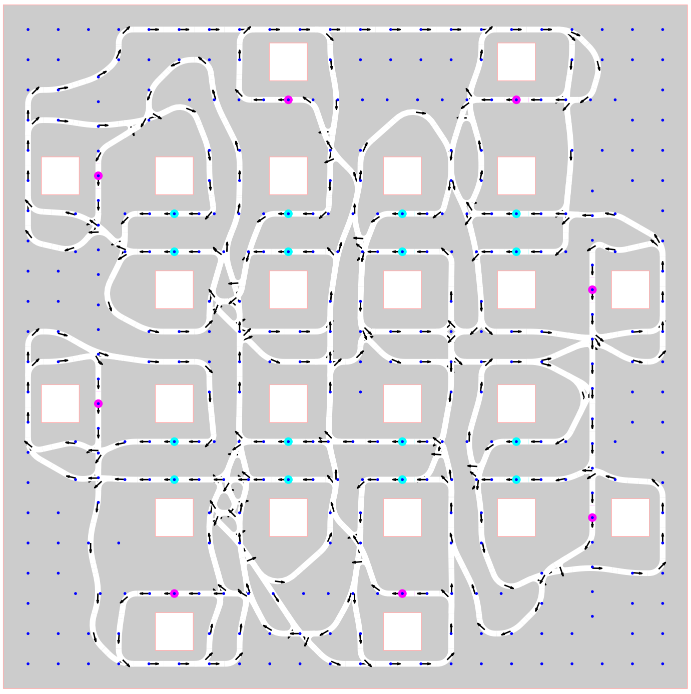


Fig. C.32: Only feasible loop solution

The influence of varying the heading of each handling point can be observed in the following images. In Figure C.33, the solutions for two different heading settings are visualised. For the left image, the headings of the handling points are not ordered and are chosen in a random manner. The layout on the right has ordered the headings such that the handling points can easier be reached via a loop, without unnecessary crossings in front of each handling point to reach the correct heading. This results in much less space used for the right layout compared to the left.

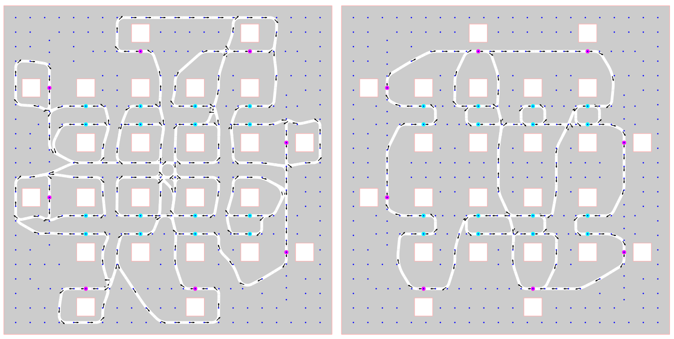


Fig. C.33: Loop results for grid based vertices

The same results are found for randomly placed vertices, as presented in Figure C.34. For this type of vertex generation, more small loops through handling points are found in the solution, compared to evenly divided placed vertices. Especially for the unordered headings, the use of forcing the algorithm to use few or many paths also results in different layouts. Even when adjusting this, the layouts remain still infeasible.

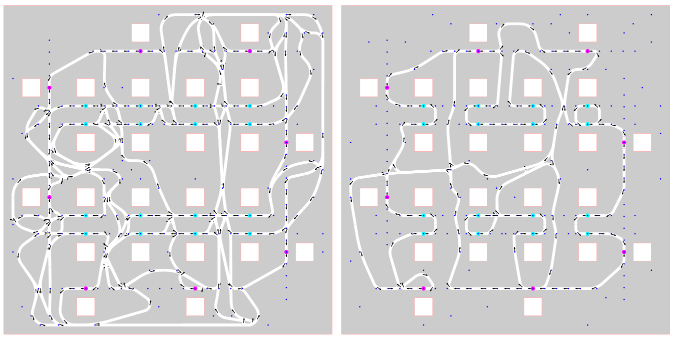


Fig. C.34: Loop results for randomly placed vertices

C.VI.3 Bypass Generator Results: Table C.12 summarizes the total costs for each different setup for the parcel instance. Obviously, forcing the algorithm to generate the least number of paths is not working for this instance and method because of the low maximum edge capacity compared to a large number of flow targets. Furthermore, ordering the heading of all handling points, again, does not result in feasible solutions. Lastly, the only feasible results are obtained when using the grid-like placed vertices.

		Fewest paths		*Most paths*	
n		**Grid**	**Random**	**Grid**	**Random**
1	**Non-ordered**	∞	∞	**4.6885**	∞
	Ordered	∞	∞	∞	∞
3	**Non-ordered**	∞	∞	5.0017	∞
	Ordered	∞	∞	∞	∞
6	**Non-ordered**	∞	∞	5.4247	∞
	Ordered	∞	∞	∞	∞
8	**Non-ordered**	∞	∞	5.6696	∞
	Ordered	∞	∞	∞	∞

TABLE C.12. Results for bypass generator method

In Figure C.35, the layouts for $n = 1$, unordered handling points, and not forcing for the least number of paths are presented. The left layout illustrates a feasible solution for the equally spaced vertices grid. Although the right solution seems to have plenty of bypasses additional to two loops through the drop-off points, still no feasibility is obtained.

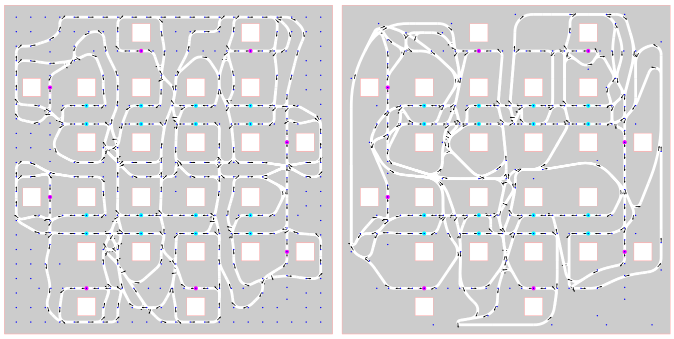


Fig. C.35: Solutions for $n = 1$ and non-ordered handling points

Figure C.36 and Figure C.37 show the second-best-performing and the worst-performing feasible solution path maps. Between those layouts, the influence of less additional bypassed (left) versus more (right) can clearly be distinguished. Not only by a larger footprint but also because of more internal connections. Therefore, the trade-off between additional path segments and redundancy is introduced again. According to the total costs, the additional paths in Figure C.37 do not compensate for the enlarged footprint enough by gaining redundancy.

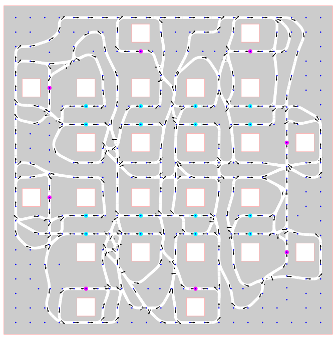


Fig. C.36: Second least costs

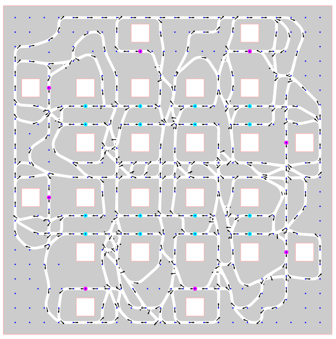


Fig. C.37: Most costs

Important is that not all possible generation setups are tested. For example, it is chosen to test n up to eight even though it can be set equal up to sixteen (leaving and arriving at each pickup point results in two times eight). If this number is increased, the total costs are increased as well due to path utilization and intersection costs and too many bypasses crossing and merging in the small corridors. Examples for both evenly spaced and randomly placed vertices for $n = 16$ are presented in Figure C.38. The solution for the grid-spaced vertices has a total cost of 4.43403. An additional path that would improve the efficiency of both layouts would be to close the drop-off points close to each other into two separate loops. Consider, for example, the left pickup point a the bottom of the layout. If an AGV needs to travel from this handling point to one of the six drop-off points of the lower cluster on the right, it has to take a huge detour. More examples of reaching drop-off points from several handling points are shown in the solutions.

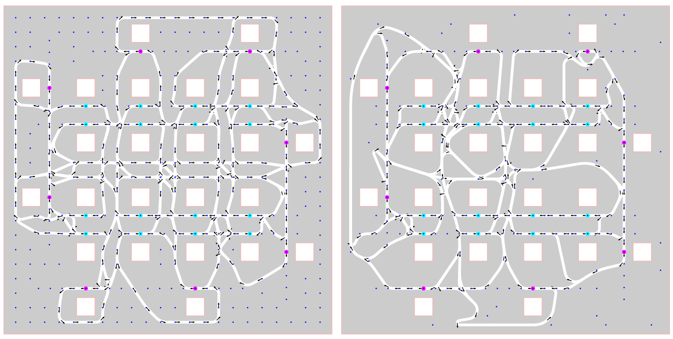


Fig. C.38: Results for $n = 16$

C.VI.4 Multiple Loop Generator Results: In the last setups for this instance, k is varied from one up to six. From Table C.13, it turns out that the k-loops method generates feasible solutions more often compared to the bypass generator. Furthermore, the total costs increase drastically for all generations without minimizing the number of paths, as presented in the two columns on the right. The best solutions are obtained using the ordered set of headings of the handling points in combination without minimizing the number of paths. At $k = 3$, the first solution becomes feasible and after increasing k even more, also more feasible solutions are generated. Apparently, $k = 3$ touches the boundary between feasible and infeasible solutions for this setup. The ordered set of handling points does not result in infeasible solutions only anymore. In contradiction, they have fewer costs compared to the exact same situation for non-ordered headings. Remarkable are the high costs for the two columns on the right. When this setup becomes feasible, the costs for path utilization and intersections are too high already because almost the entire space is used for connecting all requested neighbors.

		Fewest paths		*Most paths*	
k		**Grid**	**Random**	**Grid**	**Random**
1	**Non-ordered**	∞	∞	∞	∞
	Ordered	∞	∞	∞	∞
3	**Non-ordered**	∞	∞	6.971	7.1122
	Ordered	∞	**3.8989**	6.6971	7.0522
4	**Non-ordered**	5.3139	∞	7.3017	∞
	Ordered	4.2172	4.6454	7.4316	7.5349
6	**Non-ordered**	5.5045	∞	7.7645	∞
	Ordered	4.3562	4.615	7.8128	8.4077

TABLE C.13. Results for multiple loop generator method

The two best-performing solutions are obtained using $k = 3$ (random vertices) and $k = 4$ (grid vertices) and a minimum number of paths for ordered handling points. The results are illustrated in Figure C.39. They are similar in structure and cost distribution. All components are relatively close, only the randomly placed vertices solution uses slightly less space. Due to the weight of this component, it outperforms the solution on the right.

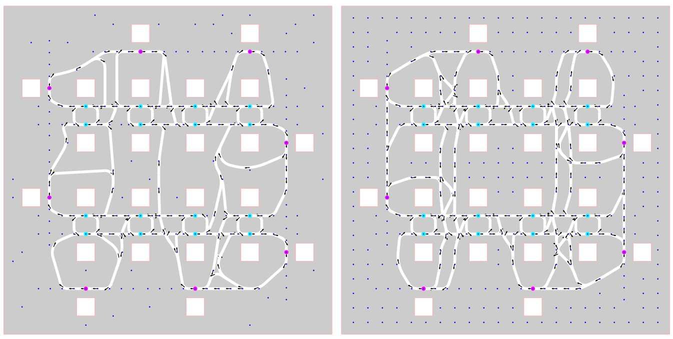


Fig. C.39: Best performing solutions, $k = 3$ (left) and $k = 4$ (right)

The layouts shown in Figure C.40 and Figure C.40 represent the solutions for $k = 6$ in a grid-like vertex graph for forcing the algorithm to the fewest number of paths on the left, and without forcing on the right. Figure C.40 uses too much space for additional paths that do not improve enough from the intersection or occupancy rate perspective. Unnecessary paths are added that do not benefit the efficiency such that the average occupancy decreases, and so results in higher costs, compared to the best-performing solution.

Figure C.41 shows the same generation setup without minimizing the paths. The paths almost cover the entire drivable space. Increasing k too much, decreases the performance and quality of the path map.

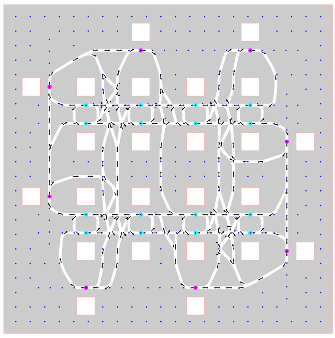


Fig. C.40: $k = 6$ few paths

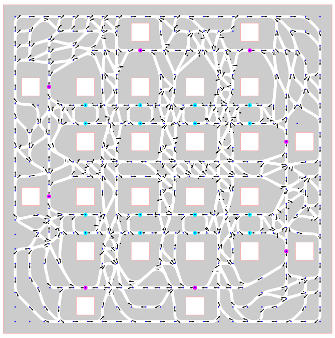


Fig. C.41: $k = 6$ many paths

C.VI.5 Changing Other Parameters: In Figure C.42, the best-performing solutions discussed in section C.VI.3 and section C.VI.4 are shown. Considering the requirements for efficient space utitilization with a high occupancy rate, the right solution would be preferred.

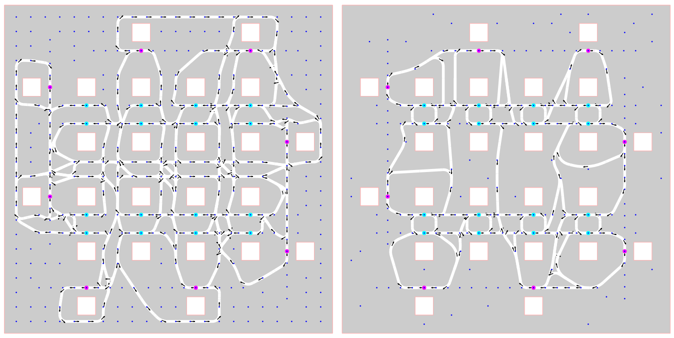


Fig. C.42: Best performing layouts for bypass and k-loops generator

To validate the expectation of two loops through the drop-off points in combination with bypasses to pickups and between the loops, an additional parameter variation is tested. A combination of the loop and the k-neighbors method is tested. First, a loop through each row of handling points is generated and connected to a loop through all pickup points, as shown in Figure C.43. All pickup points are connected based on the nearest-neighbor principle. Then, all drop-off points are connected in the same way, the last drop-off point is connected to the first pickup point.

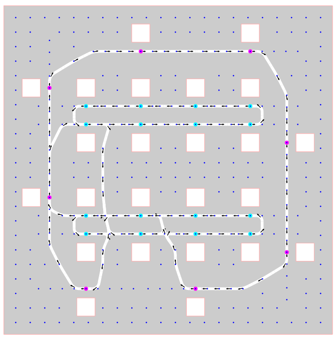


Fig. C.43: Adjusted loop

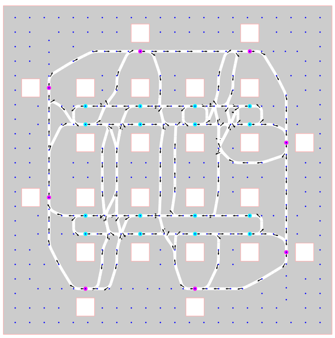


Fig. C.44: $n = 16$

After this initial loop is created, bypasses from pickups to drop-offs are added resulting in Figure C.44. From a realistic point of view, the solution looks promising. Two main loops are generated through the drop-off points close to each other. Then, many shortcuts from and towards pickup points are added with additional bypasses to change lanes. Furthermore, much less space is used and besides the loops through the drop-off points, not many intersections are generated. For this layout, $n = 16$ is used. Each pickup is connected to one of the drop-off points and eight out of 16 drop-offs are connected to a pickup location.

C.VI.6 Overview of Best and Worst Solutions Parcel Instance: The three best-performing setups and layouts are summarized in Table C.14.

Generation Setup	Score	Time [s]
Aim for min. paths k-loops, $k = 3$ Ordered handling points 500 vertices (random)	3.8989	6.275
Aim for min. paths k-loops , $k = 4$ Ordered handling points 500 vertices (grid)	4.2172	6.499
Aim for min. paths k-loops, $k = 6$ Ordered handling points 500 vertices (grid)	4.3562	4.572

TABLE C.14. Three best performing solutions parcel instance

From Table C.15, the three worst layouts can be analysed. Noticeable is that also the computation times have significantly increased compared to the airport instance (larger space and remarkably more handling points), but also compared to the three best-performing solutions. The reason for this is that all three solutions almost cover the entire drivable space with paths. This means that the path utilization, and so total, costs drastically increase with the linear programming solving time.

Generation Setup	Score	Time [s]
Do not for min. paths k-loops, $k = 6$ Ordered handling points 500 vertices (random)	8.4077	9.362
Do not for min. paths k-loops, $k = 6$ Ordered handling points 500 vertices (grid)	7.8128	18.880
Do not for min. paths k-loops, $k = 6$ Non-ordered handling points 500 vertices (grid)	7.7645	11.181

TABLE C.15. Three worst performing solutions parcel instance

C.VI.7 Exact Algorithm Layouts Parcel Instance: The exact algorithm found two feasible solutions, as presented in Figure C.45 and Figure C.46. The first obvious difference is that those layouts look cleaner and they use more space in general, especially in between the obstacles. Furthermore, many bypasses are generated from and towards drop-off points. One beneficial aspect could be that it is possible to move through all corridors that arise due to the obstacles. In the solutions of this study, this is not always the case. One huge disadvantage of the solutions of the exact algorithm is the computation time. The exact computation time is not mentioned because the execution was terminated after 10800 seconds (three hours).

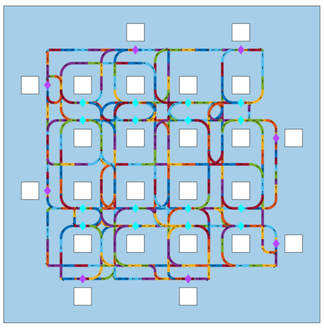


Fig. C.45: Solution 1

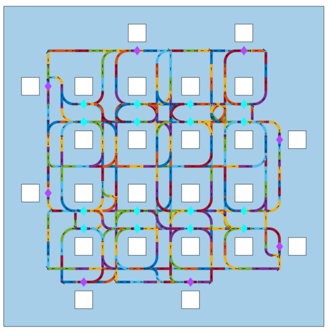


Fig. C.46: Solution 2

As presented in Table C.16, the layouts found by the exact algorithm do not differ that much in terms of the total costs. The second solution has slightly more costs due to a few more additional paths.

Solution	Cost
1	4.1331
2	4.1561

TABLE C.16. Score for comparison layouts

C.VI.8 Comparison Proposed and Exact Algorithm Parcel Instance: The partial costs per component of the cost function are shown in Table C.17.

	This study	Exact algorithm
Total cost	3.8989	4.1331
Path utilization cost	λ_1 0.2997	λ_1 0.1261
Path intersection cost	λ_2 0.3025	λ_2 0.1327
Occupancy cost	λ_3 0.4195	λ_3 0.6740
Redundancy cost	$\lambda_4 \sim 0.0$	λ_4 0.0
Computation time	6.275 [s]	5400 + [s]

TABLE C.17. Comparison between cost components

The undirected graphs used for both solutions are presented in Figure C.47. The first noticeable aspect is that the graph used for the exact algorithm is much denser, for example, at the horizontal corridor in the middle of the environment. The reason for this is that the exact algorithm generates a more detailed and developed grid. For example, having many vertices around obstacles and handling points such that the paths can be aligned strictly around them. This, in combination with the check and adjustments for hindering paths, is also a reason that the exact algorithm produces more clean and realistic acceptable path maps.

The path utilization costs for the exact algorithm are less because of the total amount of possible paths. Due to the denser graph, more possible paths could be drawn in space. Thus even though the path map for this layout uses much less space, it does not outperform the total possible path length for the exact algorithm, which uses the space much more efficiently around and in between obstacles.

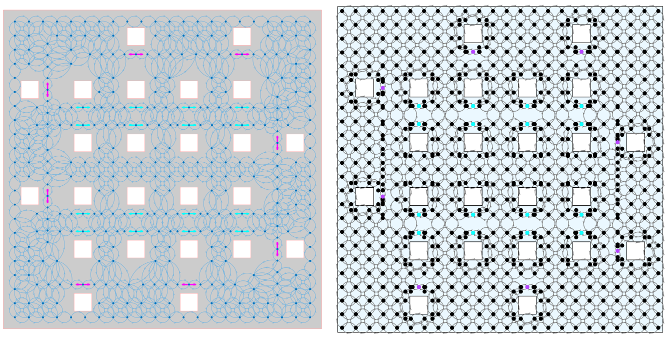


Fig. C.47: Used graph for this study (left) and exact algorithm (right)

Regarding path utilization and intersections, the exact algorithm outperforms the proposed framework. Again, an important reason for this is the graph. Especially since the best-performing solution for this layout uses randomly generated vertices, not the graph shown in the left image of Figure C.47. An example undirected graph generated via randomly placed vertices is shown in Figure C.48, note that due to its randomness it does not exactly match the graph used for the best-performing solution. Obviously, this graph contains much fewer vertices and edges, which is the reason why, if the total path length does not differ too much, the costs for path utilization are higher. The same holds for the intersections, the number of intersections does not differ that much, but the number of used edges does.

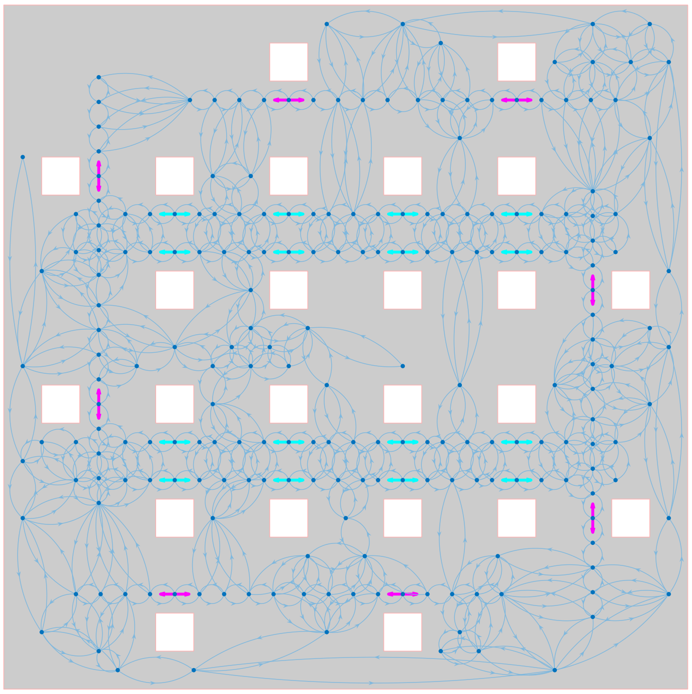


Fig. C.48: Undirected graph for random vertices

The costs for redundancy are acceptable for both layouts, at least two paths from and to almost each handling point exist. However, there are some considerations. For example, the pickup point the closest to the lower left corner in Figure C.42 is partly only reachable by one path, leaving the lower drop-off points loop. Considering this pickup point from the perspective of all handling points, several different paths exist. The cost for this handling point in terms of redundancy, since this is only one path to a handling point out of 8 pickup points $\times$ 18 drop-of points $\times$ 2 for both directions $\times$ 0.25 (due to the weight factor), equals a number ~ 0.

The component for which the proposed algorithm layout outperforms the exact algorithm is the occupancy rate. Due to fewer paths and path length, the average occupancy per path segment is higher, emphasized by the weight factor.

C.VI.9 Experts Analysis: When analyzing both solutions in Figure C.49, it can be stated that the right solution looks more realistic in terms of paths crossing, merging, and diverging with the dimensions of AGVs in mind. It seems that more realistic corners are used within the layout as if an additional smoothing phase is included. An advantage of the left layout is the idea of creating two loops through the handling points. Adding bypasses and shortcuts to a loop of pickups or individual pickup points is a strategy often used in reality. This, in combination with all of the additional bypasses to the pickup points, shown in the right layout, could be a great way of starting this layout. However, both layouts are too complicated. When an easy-to-model layout (as in, only horizontal and vertical corridors) is given with high target flows, the easiest way of meeting the requirements here would be to draw horizontal and vertical paths through the handling points such that a grid type of path map is drawn. This ensures that all AGVs can drive to and from each handling point relatively easily and due to the many options for diverging and merging the layout performs well in terms of redundancy as well. Both solutions give a well thought and calculated indication with theoretical feasibility guarantees, this can be used to analyse possibilities within the given layout.

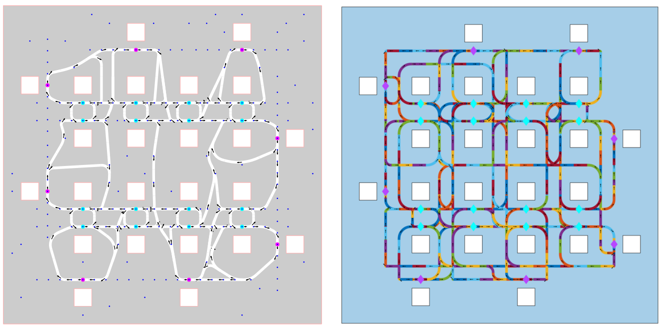


Fig. C.49: Best performing solutions parcel instance

C.VII Realistic Instance

Vanderlande designed a path map for this realistic instance. Here, additional types of handling points are included. The layout is visualised in Figure C.50, for which the pickup and drop-off points are colored in purple (magenta) and cyan respectively. Additionally, charging points are visualised in red, and one parking spot is colored in green.

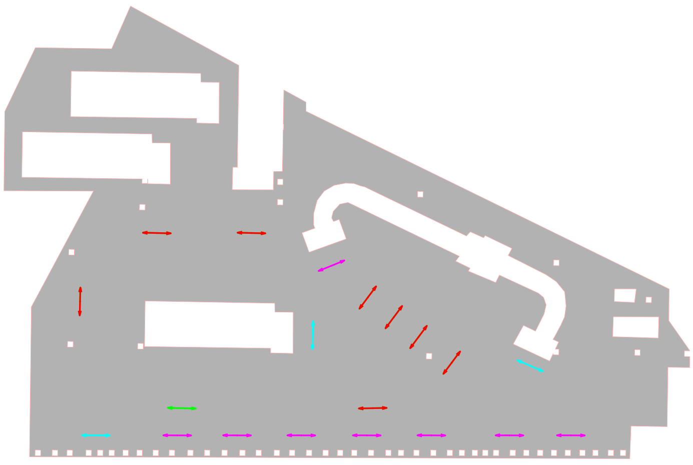


Fig. C.50: Realistic layout

The goal for this instance is to pick up items at the purple pickup points in line at the bottom of the layout at first. Second, all picked-up items should go through a screening line, starting at the drop-off point at the right corner bottom of the layout. This obstacle represents a straight screening line with two corners. In front of the entrance, there is a drop-off point located, and a pickup point is located at its exit. In between those two handling points, four charging spots are located. Then, when an item has successfully passed the screening line, it should be moved to the exit location at the drop-off point at the center of the layout, just in front of the obstacle in the middle. Above the obstacle next to the drop-off exit, three more charging points are located. Important is to mention here is the paths leading to charging points ideally have multiple in and outgoing paths, otherwise, this entire path is blocked if one or more AGVs are charging. Lastly, the drop-off point at the left lower corner is located to manually pick items from the system, items can be dropped op at this place. This is called a clearing or manual encoding drop-off. Accordingly, the pickup point next to it is corresponding to this drop-off point. Finally, the parking spot, marked in green, is located above this pickup point.

C.VII.1 Parameter Explanation: The flow targets leave from each pickup point to each drop-off, charging, and parking point. The requested flow equals one unit per hour with a maximum edge capacity of 100 units per hour. Furthermore, the number of generated vertices is set equal to 200 and the headings are fixed as well. Due to the relatively available space, it is expected that forcing the algorithm to use the fewest amount of path segments performs better. However, this parameter is kept variable to analyse the resulting performances.

Table C.18 summarizes the parameters used in the generation setup. In the last column, the weight factors for the analysis based on the cost function are set. For this instance, the space utilization, or the footprint, is most important together with redundancy. Space utilization is important since there is not much drivable space available and redundancy is of great importance due to the many charging or parking spots. When an AGV is located here, the system can still work around it. Furthermore, the path intersection and occupancy rate are considered to be less important, as expressed by the weights.

Instance	Flow targets	L	$e_{\max}$	Weights
Realistic	1 $[units/hour]$	1.3 $[m]$	100 $[units/hour]$	λ_1 = 5 λ_2 = 1 λ_3 = 0.25 λ_4 = 5

TABLE C.18. Generation setup for realistic instance

For this instance, there is no data available in order to have a comparison of total costs. However, it is still decided to compute the cost estimation for several generations such that an indication of the performance is known. Besides that, there will be an expert analysis at the end of this section in which the scores of the best-performing layouts are compared, those validations can be compared after all.

C.VII.2 Loop Generator Results: The solutions for the loop generator are visualised in Figure C.51 with a total cost equal to 3.7259 and 3.6608. Both layouts seem to be well-performing solutions. A straight path through all pickups at the bottom of the layout is complemented by a path toward the drop-off point in front of the screening line. Then, there are options to switch lanes to charging points or drive toward the pickup point at the end of the screening line. From here, the item can be dropped at the exit drop-off point, and the AGV can decide whether to go back to the pickup points or the three other charging points on the left side of the layout.

A remark on both solutions is the lack of a shortcut from the exit drop-off location to any other pickup point of the bottom pickup point line. Furthermore, in terms of redundancy, both layouts perform badly due to one possible path only for entering or leaving the three charging points on the left. This means there is no shorter option to reach the drop-off point at the left bottom corner and then drive via three charging points, which could be occupied as well.

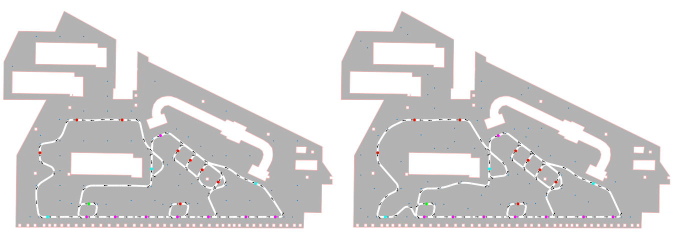


Fig. C.51: Loop generator solutions realistic layout

C.VII.3 Bypass Generator Results: To improve both loops, especially in terms of redundancy, additional bypasses are generated. The generations are done with both using the fewest and most paths possible. In Table C.19, the scores are summarized for all different setups.

	Fewest paths		*Most paths*	
n	**Grid**	**Random**	**Grid**	**Random**
2	3.2789	**2.7885**	3.0886	3.5109
4	3.1980	2.8204	3.1356	3.5628
6	3.3487	2.8967	3.3351	3.8542
8	3.3904	2.9851	3.4456	3.8796
10	3.0339	2.9977	3.5720	3.9309
12	3.0361	3.1389	3.7979	4.0461

TABLE C.19. Results for bypass generator for realistic instance

Regarding the scores, the randomly placed vertices with path minimization performs the best. Considering those scores (second column from the left), it seems that increasing the number of bypasses only results in the path utilization costs overtaking the redundancy profits, resulting in higher costs.

For the grid-spaced vertices, the increasing value for n sometimes results in higher but also lower total costs. A reason for this could be that at some configurations, additional paths are generated that do not benefit other components enough to overcome the footprint costs. The randomly placed vertices while aiming for as many paths as possible perform the worst by far. The total costs increase each time the number of bypasses increases as well, probably there are graph is not dense enough due to the dimensions of the instance. Leading to many unrealistic and unnecessary paths that use lots of space.

Two solutions for aiming for the least number of paths while using evenly divided-spaced vertices are shown below. In Figure C.52 the number of bypasses is set equal to four and in Figure C.53 ten bypasses are used. According to Table C.19, the total costs for both solutions do not differ too much. However, when looking at the left layout more closely, some shortcomings compared to Figure C.53 are clearly visible. First of all, the left solution only has one entrance to the three charging points on the left, this is solved when increasing n up to ten. Second, when leaving the drop-off point in front of the exit at the center of the layout, the left solution only has one path towards the straight line through all bottom pickup points. The right solution has two, one path is added halfway, which is beneficial for the overall performance. One important path is missing for both solutions, the connection between the center drop-off to one of the groups with charging points. For both maps, a huge detour is required to reach one of those.

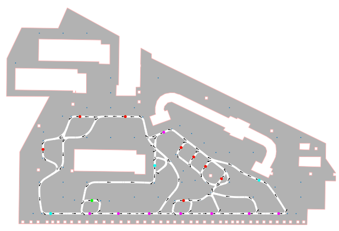


Fig. C.52: Grid, $n = 4$

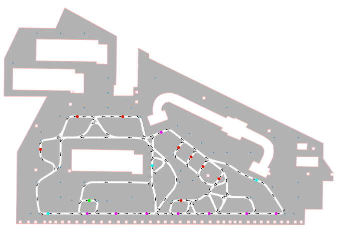


Fig. C.53: Grid, $n = 10$

Figure C.54 and Figure C.55 visualise two solutions for using randomly placed vertices while aiming for the least number of paths in the layout. When comparing both scores, $n = 2$ (left) outperforms the right solution in which eight bypasses are requested. However, for both solutions, the important connection from the central drop-off point directed towards the chain of pickup points at the bottom of the layout is missing. Therefore, in terms of scores, both solutions perform well, but in reality, this missing aspect would end up in terrible performances. A promising aspect of both solutions is the additional path that results in the bypass around the drop-off point at the left bottom corner, towards the chain of pickup points.

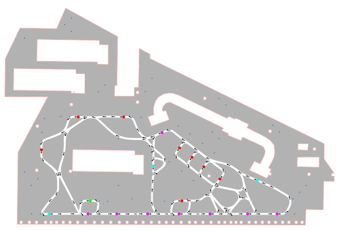


Fig. C.54: Random, $n = 2$

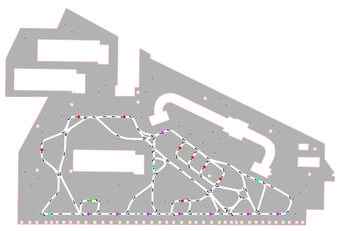


Fig. C.55: Random, $n = 8$

Now the third and fourth columns of Table C.19 and their visualisations are analysed. Overall, all costs are higher compared to generating the fewest number of paths. In Figure C.56 and Figure C.57 the visualisations for the general loop with two additional bypasses are shown. Obviously, much more paths are already used when only two additional bypasses are requested. On top of that, it shows that the chain of pickup points is not connected anymore. Therefore, the path from the first pickup point in the chain toward the screening line is much less efficient. Furthermore, too many unnecessary diverges, merges, and crossings are added. For example, in the left figure, many crossings and small loops between the pickup line and the charging points are added. It does not make sense to charge after picking up items from this chain.

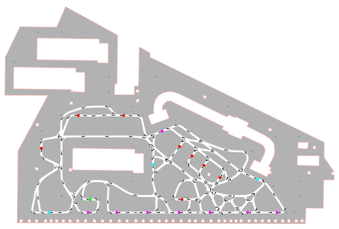


Fig. C.56: Grid, $n = 2$

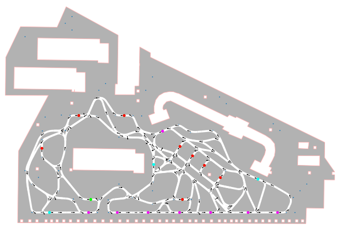


Fig. C.57: Random, $n = 2$

Increasing the number of bypasses even further results in worse overall performances for both setups. From Figure C.58 and Figure C.59 the solutions with the largest costs for both grid or randomly placed vertices can be observed. Even though the redundancy for both layouts is drastically improved, too many paths are drawn in the drivable space to consider those results as suitable.

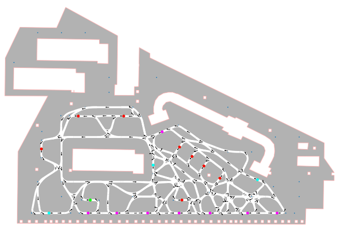


Fig. C.58: Grid, $n = 12$

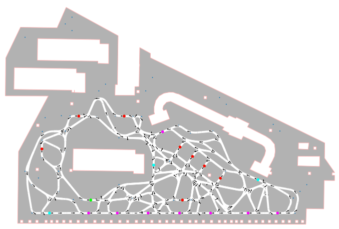


Fig. C.59: Random, $n = 12$

C.VII.4 Multiple Loop Generator Results: Some handling points are located very close to one another, and some can be seen as outliers. Therefore, once the path map is strongly connected, k should be increased a lot. With k being relatively large, the path map will be entirely filled by paths.

If one connects handling points only based on distance in this situation, logically chosen connections will be made. For example, before the connection between the drop-off point and pickup point in front of the screening line is made, k should be enlarged a lot due to all of the nearby other handling points that are connected first.

Almost all solutions are infeasible, as shown in Table C.20. Once the solutions turn feasible, the costs are large as well due to too many paths generated in the drivable space. An important note for those generation settings is that the solutions for randomly generated vertices can strongly differ per generation in terms of feasibility.

	Fewest paths		*Most paths*	
k	**Grid**	**Random**	**Grid**	**Random**
1	∞	∞	∞	∞
2	∞	∞	∞	∞
3	∞	∞	∞	∞
4	∞	∞	∞	5.1177
5	∞	∞	∞	5.0590
6	3.8851	4.2183	5.1472	4.7985

TABLE C.20. Results for k-loops generator for realistic instance

Two feasible solutions obtained after using $k = 6$ and aiming for the least number of paths are illustrated in Figure C.60 and Figure C.61. The number of intersections introduced to the layout by this method is too large and in combination with a too large footprint makes this solution not ideal. The same goes for the randomly placed vertices, as shown in the right layout.

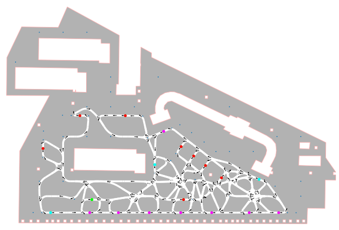


Fig. C.60: Grid, $k = 6$

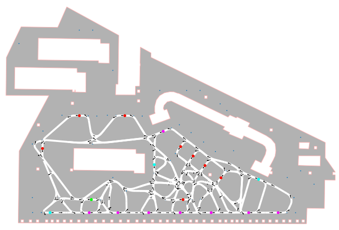


Fig. C.61: Random, $k = 6$

Figure C.62 and Figure C.63 show the difference in space covered by paths needed before turning the infeasible solutions feasible. In the left layout, the not strongly connected network is clearly visible at several handling points. The right solution shows that almost all space is covered by paths.

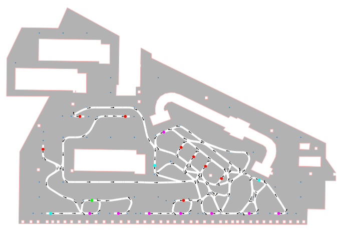


Fig. C.62: Grid, $k = 1$

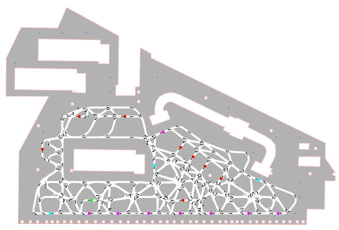


Fig. C.63: Grid, $k = 6$

In Figure C.64 and Figure C.65 the same situations for randomly generated vertices are presented. It seems that too many unnecessary paths are added without the idea of meeting the given requirements.

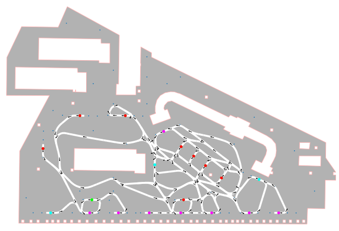


Fig. C.64: Random, $k = 1$

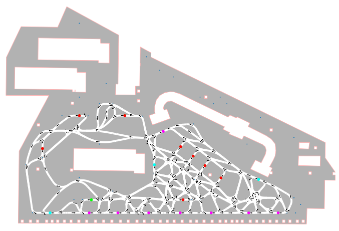


Fig. C.65: Random, $k = 6$

C.VII.5 Overview of Best and Worst Solutions Realistic Instance: Based on the previously discussed layouts, it is shown that the best-performing solutions are obtained via the bypass generator, and the worst solutions via the multiple-loop generator. The three best-performing solutions, according to the cost function, are presented in Table C.21. The three worst-performing solutions are shown in Table C.22, all generated with the k-loops method.

Generation Setup	Score	Time [s]
Aim for min. paths Bypass, $n = 2$ 200 vertices (random)	2.7885	3.454
Aim for min. paths Bypass, $n = 4$ 200 vertices (random)	2.8204	3.525
Aim for min. paths Bypass, $n = 6$ 200 vertices (random)	2.8967	3.507

TABLE C.21. Three least cost solutions for realistic layout

Generation Setup	Score	Time [s]
Do not aim for min. paths k-loops, $k = 6$ 200 vertices (grid)	5.1472	5.515
Do not aim for min. paths k-loops, $k = 4$ 200 vertices (random)	5.1177	5.477
Do not aim for min. paths k-loops, $k = 5$ 200 vertices (random)	5.0590	4.444

TABLE C.22. Three largest cost solutions for realistic layout

C.VII.6 Expert Analysis: The main goal for this instance is to find a solution with the smallest footprint that finds the highest, but also logical, throughput. Relating throughput to the given cost components, it can be stated that the layouts should be redundant enough at the correct locations. When considering the two groups of charging points (one at one the left side of the layout, and one in between the pickup and drop-off point in front of the screening), it is strongly recommended to have (at least) two paths going in and two paths going out. This makes it possible to keep the flow going when AGVs are charging. The difference between relevant redundancy and overall redundancy is not taken into account in the currently used cost computation. Therefore, some well-performing solutions according to the cost function, are not as promising from a realistic point of view. Furthermore, the cost function does not include the important paths for the particular instance. For example, one of the most important paths is the connection from the exit drop-off location, at the center of the layout, to the chain of pickups at the bottom. Looking at the solution with the least costs, this connection exists, but with a huge detour. Therefore, this solution is not preferable in realistic layouts. The most promising solution for the shown generation settings is presented in Figure C.66.

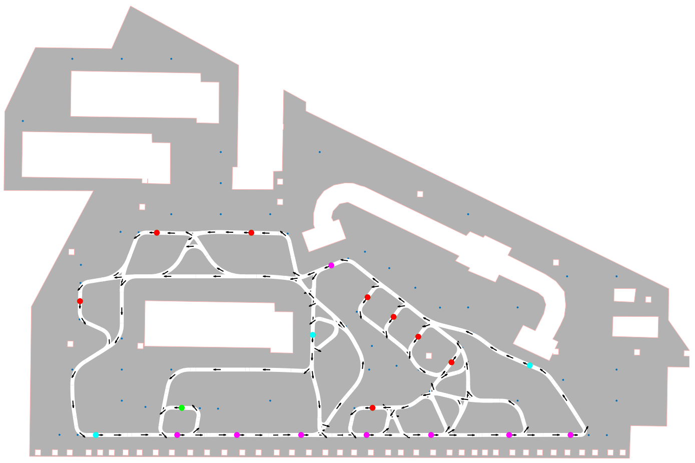


Fig. C.66: Most promising solution

For this generation, the bypass method for an equally spaced grid with n equal to ten is used. This layout contains realistic paths for which, at some points, the necessary redundancy is added. However, two important bypasses are missing still. First, when leaving the cluster of three charging points at the left half of the layout, AGVs should always move via the drop-off point at the left bottom corner. A bypass would solve this. Second, the cluster of four charging points in between the screening line misses a bypass towards the chain of pickups at the bottom such that charges AGVs can immediately pick up items and join the cycle of flow again. Also, the path connecting the clearing or manual encoding pickup point to the centrally located drop-off point is of great importance because now the manually picked items can be sent back into the system.