CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Framework for multi-agent path finding

A framework for multi-agent path finding with focus on transport systems

Master's thesis in Computer science and engineering

Andreas Kuszli & Jesper Åberg

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Framework for multi-agent path finding

A framework for multi-agent path finding with focus on transport systems

Andreas Kuszli & Jesper Åberg

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

A Chalmers University of Technology Master's thesis
A framework for multi-agent path finding with focus on transport systems

© Andreas Kuszli & Jesper Åberg, 2021.

Supervisor: Elad Schiller, Department of Computer Science and Engineering
Examiner: Nir Piterman, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Typeset in LaTeX
Gothenburg, Sweden 2021

A Chalmers University of Technology Master's thesis
A framework for multi-agent path finding with focus on transport systems
Andreas Kuszli & Jesper Åberg
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

It is desirable to apply multi agent path finding (MAPF) algorithms to vehicles in transport systems, as it has the potential to reduce transit time and fuel consumption. However, the application is hampered by the high level of abstraction that these problems are typically solved at. This work contributes to the field by introducing a method that lowers the level of abstraction as a post processing phase. We also evaluate a recently developed algorithm that considers mobile agents in continuous time with discrete speeds, and measure its performance when compared to other algorithms. Further, we provide a framework containing these algorithms and the tools required to evaluate and develop them.

Keywords: Computer, science, computer science, engineering, project, thesis, Multi agent path finding, Framework.

Acknowledgements

We would like to express our gratitude to our supervisor Elad Schiller, who has supported this work with great knowledge and guidance. We would also like to thank Nir Pieterman for being our examiner and accepting this project. Further, we thank Johan Gerdin and Andreas Rosenfeld, who both provided us with valuable discussions and contributions to the code used in this work.

Andreas Kuszli & Jesper Åberg, Gothenburg, January 2021

Contents

List of Figures

List of Tables

1
Introduction

Multi-agent pathfinding (MAPF) is the problem of finding a set of paths for a set of agents. A solution to a MAPF problem is a set of collision free paths where each path takes an agent from a starting location to a goal location. This problem can be applied to many areas where it is desirable to have multiple autonomous agents such as warehouse robots or vehicles on the road. Successfully applying MAPF to these areas would reduce fuel consumption and transit times, as vehicles would travel the shortest possible route. However, it is NP-hard to solve MAPF optimally, meaning that it is very hard to solve for large numbers of agents[19]. In this work, we focus on vehicles in transport systems, such as cars or trucks on the road.

When all the properties of physical agents are considered, it is referred to as the target system. No MAPF solver considers all the constraints of the target system. Instead, the problem is solved in an abstract model of the target system. For example, time might be assumed to be discrete rather than continuous and agents are assumed to have unbounded acceleration. Problems arise when applying the produced plans to mobile agents in the target system, either collisions occur that were not detected in the solver or agents are unable to execute their paths due to physical constraints that were not accounted for. One work that attempts to mitigate this is the MAPF-POST algorithm [10]. Their proposed solution is to solve the MAPF problem on a high level of abstraction and then model the kinematic constraints in a post-processing phase. This method has some limitations, graphs can not contain crossing edges and the length of all edges must be above a certain length that depends on agent radius. Another limitation is that it does not consider the bounded acceleration of mobile agents at the target system. Another approach is to solve the problem at a lower level of abstraction right away, such as CCBS [1] and DS-CBS [9]. However, solving problems at lower levels of abstraction requires the solver to model more details of the target system which might increase the runtime required to find a solution.

The long runtime required to solve MAPF problems limits the use of these algorithms. MA-CBS [18] is an attempt to mitigate this problem. MA-CBS works by detecting strongly coupled agents, that is, agents who often collide. Strongly coupled agents are combined into meta-agents who are solved independently, this reduces the runtime required to find a solution.

This work revolves around the evaluation of a new algorithm that lowers the level of abstraction by considering agents with bounded acceleration[9]. We are interested in how this algorithm performs when compared to algorithms that assume a higher level of abstraction. We also develop our own post processing method that transforms a solution produced with any abstract model into a solution at the target system.

1.1 Related work

The work most related to ours is that of Gerdin [9] from which we borrow models and algorithms. Earlier work in the area includes Petig et al. [12], which offers an algorithm for lane changing, as well as Ekenstedt et al. [3], which offers a protocol for intersection crossing.

1.2 Our contributions

We contribute to the field of applied MAPF by providing a framework for future developments. The main feature of this framework is the inclusion of three different MAPF solving algorithms, working on different levels of abstraction. We also contribute with a novel method that transforms a solution produced with any abstract model into a solution at the target system. While this method is not optimal under all circumstances, it is the only method we know of that can perform this transformation. Perhaps the most significant contribution of this work is a thorough, systematic evaluation of these different models, and how they are affected by a wide array of circumstances and parameters.

2

Background knowledge

In this chapter, the background knowledge required to understand this work is presented. The chapter starts with the introduction of MAPF problems and the different levels of abstraction that are considered. Then, MAPF-POST is introduced, an algorithm that converts discrete time solutions to continuous time with constant speed. Finally, suboptimal search configurations and bounded delay tolerance are presented.

2.1 Multi Agent Path Finding

MAPF is the problem of finding a set of paths over a graph $\mathsf{G} = \{\mathsf{V}, \mathsf{E}\}$ for a set of agents $\mathsf{A} = \{\{\mathsf{s}_0, \mathsf{g}_0\}, \ldots, \{\mathsf{s}_n, \mathsf{g}_n\}\}$ from their respective starting vertices (s_x) to their respective goal vertices (g_x). A single path contains all the necessary information an agent needs to traverse the graph from its start vertex to its goal vertex. A valid solution to this problem will have one collision free path per agent. An optimal solution to this problem is a solution where the cost of the solution is as small as possible for the provided problem instance. The cost of a solution can be defined in several ways, the two types of costs considered in this work is the sum of individual costs (SIC) and the cost of the most expensive path (makespan). Solving a MAPF problem optimally is NP-hard [19].

The solving algorithms considered in this work are all based on CBS [18], which works in two levels, one called high level and one called low level. In the low level, paths for each agent are generated individually as instances of single agent path finding problems, typically with some variant of A*. Agents must respect a set of constraints which prevents them from performing certain actions, such as moving between or waiting at certain vertices at specific times. When all paths have been generated, they are examined in search for collisions. If no collisions are found, the generated paths are a valid solution and is returned. If a collision is found, constraints are generated, these prevent the collision from occurring again. Constraints are also used to perform a search in the high level. For every constraint, a high level node is added to a search tree, called the constraint tree. When a high level node is evaluated, the low level is invoked with a set of constraints. The set of constraints is generated using the constraints of the high level node and all the parents of that node. The cost of a high level node must be calculated prior to being added to the constraint tree, as the cost is being used in the search. The node with the lowest cost is always opened next, ensuring that the first found valid solution is an optimal solution.

2.2 Models

There are several different MAPF solvers that take different physical properties of agents into consideration. In other words, they assume different levels of abstraction compared to the target system. In this section we present the models that were considered for this work and proposed by Gerdin [9].

2.2.1 Discrete time model

In the Discrete Time (DT)model, all agents progress through the graph in a synchronized manner. The time an agent would need to traverse an edge is not considered, nor is their physical size. Agent paths are defined as a set of locations $\{location_0, \ldots, location_n\}$ that describe which vertices an agent will travel through and then the agent visits those locations. The algorithm used in this model is CBS [18].

2.2.2 Continuous time model

The key differences between the Continuous Time (CT) and the DT models are that in the CT model, agents can wait a continuous amount of time before moving between vertices and the movement between vertices is continuous. Collision detection also considers the physical size of agents. Agent actions in the CT model are defined by $\{location, time\}$ pairs which forms an itinerary $\{location_0, time_0, \ldots, location_n, time_n\}$ to represent agent paths. The time it takes to move between vertices is defined as a cost that is defined per edge, this cost can for example be the length of an edge.

An algorithm called Continuous Conflict Based Search (CCBS) [1] is used to solve problems in the CT model. The algorithm extends CBS by introducing Safe Interval Path Planning (SIPP) [13] as a low-level algorithm to find single agent paths and avoid conflicts. Constraints are defined as unsafe intervals on edges for move actions or vertices for wait actions.

2.2.3 Discrete speeds model

The Continuous Time Discrete Speeds (CTDS) model, as proposed and developed by Gerdin [9], extends the CT model by relaxing the assumption that agents move with constant speed. Agents in the CTDS can travel between vertices at a predefined set of speeds, also modelling acceleration. To accommodate this addition, the path representation is changed to be an itinerary of $\{location, time, speed\}$ sets. The time it takes for an agent a to traverse an edge $e = \{v_1, v_2\}$ is now a function of the length of e and the speeds that a has at the vertices v_1 and v_2.

2.3 MAPF-POST

MAPF-POST is an algorithm that transforms a solution in the DT model to a solution in the CT model [10]. The algorithm considers the physical size and bounded velocity of agents, but not the acceleration constraints. This method is applied after

a solution has been found by a solving algorithm. It is assumed that the agents are able to compensate for the lack of planned acceleration. This section explains in depth how the algorithm works.

2.3.1 Temporal Plan Graph

First, a data structure called Temporal Plan Graph (TPG) is constructed using a complete solution. A TPG is a directed, acyclic graph $\mathsf{G} = \{\mathsf{V}, \mathsf{E}\}$ where each vertex $\mathsf{v} \in \mathsf{V}$ corresponds to the event of an agent reaching a location. Each edge $\{\mathsf{v}, \mathsf{v}'\} \in \mathsf{V}$ represents a temporal precedence, indicating that v occurs no later than v'. In other words, each vertex v corresponds to an agent a reaching a location. Each edge directed to v indicates that some agent must reach some vertex before a is allowed to reach v.

There are two types of edges in a TPG. A Type 1 edge is an edge $\mathsf{e} \in \{\mathsf{v}, \mathsf{v}'\}$ where both v and v' correspond to events in the path of one agent. In a Type 2 edge $\mathsf{e} = \{\mathsf{v}, \mathsf{v}'\}$, the events do not belong to the same agent. The edges are used to encode temporal dependencies between events. The TPG does not define at what time an agent reaches a location, it only defines in what order events must transpire. The figures below show an example problem and the corresponding TPG.

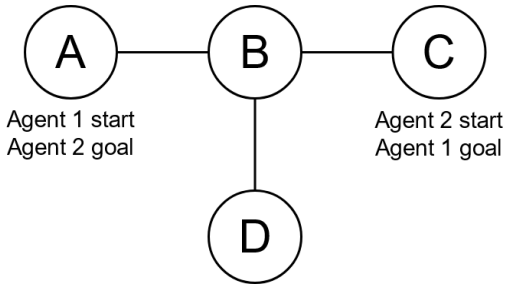

Figure 2.1: Graph structure for our running example.

	t=1	t=2	t=3	t=4	t=5	t=6
Agent 1	A	A	B	C	C	C
Agent 2	C	B	D	D	B	A

Table 2.1: Paths that the agents traverse on the running example.

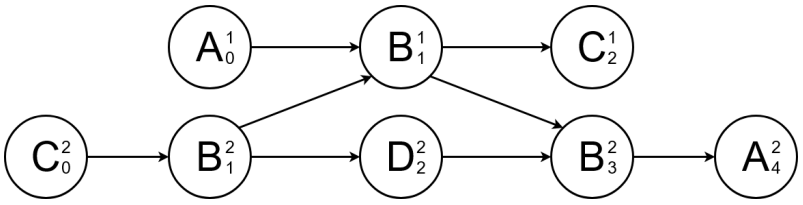

Figure 2.2: Structure of a TPG. Event superscript indicates which agent it belongs to while subscript shows the events index in the path.

2.3.2 Augmenting the TPG

After the TPG has been generated, it is modified by inserting extra vertices called safety markers that correspond to new locations. The purpose of safety markers is to provide a guaranteed minimum distance between agents that travel through the same vertex. For each type 1 edge $\mathsf{e} = \{\mathsf{v}, \mathsf{v}'\}$ two new vertices are inserted. A new edge between v and the first safety marker is created, another edge connecting the two safety markers and finally another edge from the second safety marker to v' is created. The original edge e is removed. The safety markers should be at least some distance away from their neighbouring original vertices as to avoid collisions, the distance depends on the radius of agents and the angle between edges and should be provided by the user.

After the safety markers have been created, all type 2 edges need to be redirected. For each type 2 edge $\mathsf{e} = \{\mathsf{v}, \mathsf{v}'\}$, move the edge so that it connects the vertex following v to the vertex preceding v'. The result of adding safety markers can be seen in figure 2.3.

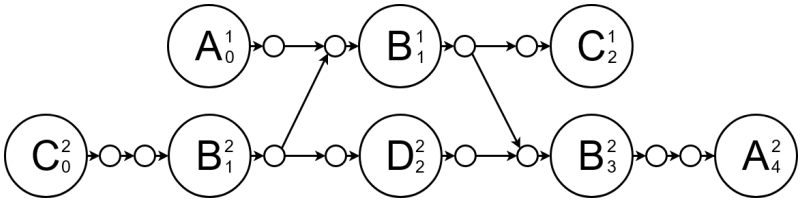

Figure 2.3: Structure of an augmented TPG.

2.3.3 Encoding Kinematic Constraints

Quantitative information is now associated with the edges, transforming it to a data structure called Simple Temporal Network (STN). Every edge $\mathsf{e} = \{v, v'\}$ is associated with temporal constraints called Upper Bound and Lower Bound, annotated UB(e) and LB(e). These bounds define a span after v in which v' must be scheduled. Two new vertices $\mathsf{X_S}$ and $\mathsf{X_F}$ are added, $\mathsf{X_S}$ represents the start event and $\mathsf{X_F}$ represents the finish event. The start event represents the first point in time at which agents are allowed to perform actions, and the finish event represents reaching the goal location. The edges connected to $\mathsf{X_S}$ have bounds [0,0] while the edges connected to $\mathsf{X_F}$ have bounds $[0, \infty]$.

Since all edges now define a temporal precedence all Lower Bounds must be no smaller than 0 while all Upper Bounds can be infinitely large. Because of the bounded velocity of agents, tighter Lower Bounds can be assigned to all type 1 edges, as it always takes more than 0 time units to perform an action. For a type 1 edge $\mathsf{e} = \{\mathsf{v}, \mathsf{v}'\}$ the LB can be assigned to be equal to the time it takes the agent to traverse the distance between v and v' at its top speed.

The STN is converted to a distance graph after all bounds have been found. The difference between an STN and a distance graph is that it is no longer directed, as back facing edges are added. A back facing edge $\mathsf{e} = \{\mathsf{v}, \mathsf{v}'\}$ is one where event v must be scheduled later than v'. The forward facing edges are associated with the

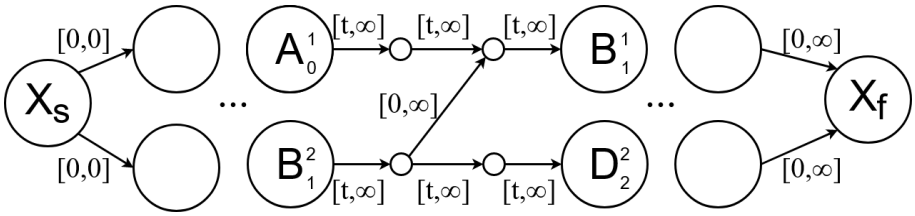

Figure 2.4: Graph after all processing has been applied.

positive Upper Bound of the STN edges while the back facing edges are associated with the negative Lower Bound of the STN edges.

After the distance graph has been generated the graph can be optimized for minimal makespan or SIC by using linear programming or graph-based optimization methods, for example the Bellman-Ford algorithm.

2.4 Low Level Heuristics

Heuristics in the context of A* is a method that approximates the cost from any vertex to the goal vertex. As described by Sadikov et al. [16], using pessimistic or optimistic heuristics in single agent path finding will produce different solutions. For an optimal search the heuristics must be admissible, but using a pessimistic heuristic reduces the runtime of the algorithm. An admissible heuristic is characterized by never overestimating the actual cost while a pessimistic heuristic might overestimate the cost. In this work, pessimistic heuristics are denoted by $b \cdot$ admissible where b is some constant bias. Pessimistic heuristics are calculated by first calculating the admissible heuristic and then scaling with the bias factor.

3
Heuristics and methods

In this chapter, the methods developed in this work are explained. The first method is used to transform solutions to the target system. We then present how suboptimal search methods can be used to reduce the runtime of the algorithms.

3.1 Enhanced MAPF-POST heuristic

The application of MAPF algorithms to mobile agents is inhibited by the high level of abstraction that MAPF solving algorithms typically assume. Solving MAPF problems at the target system is prohibitively hard to do. Enhanced MAPF-POST adds the details of the target system as a post processing phase and is compatible with any MAPF solving algorithm.

We propose a heuristic which takes a solution produced by any of the proposed models as input and transforms it to the target system. The resulting solution represents agent position, velocity and acceleration with continuous values. Collisions between agents are prevented by using temporal dependencies, represented with an augmented TPG structure.

For this method, the structure of the TPG was altered slightly. The original version does not allow for agents to have the same goal vertex. To allow this, a vertex is added at the end of every path in the TPG. This vertex does not correspond to a physical location, however the arrival time is defined to be large enough for the agent to safely reach a distance outside of the graph without collisions.

3.1.1 Initialization

Before paths can be optimized, initial arrival times for each vertex in every path must be known. The optimization phase must start in a valid initial state, that is, all initial arrival times are feasible with the acceleration and velocity constraints imposed on the agents. The input solution can not be used for initial state as it might have been computed in a model that assumes that agents can travel instantly between vertices, which is not a valid state. The first step in the initialization phase is to create an augmented TPG, then initial arrival times must be calculated for every vertex. The procedure is described in the following paragraphs and in Heuristic 1.

During the initiation, agents are presumed to travel with a constant speed v that is provided by the user. The speed must be low enough that every agent is able to reach an average speed of v for every edge in their path. If this condition is not met,

Heuristic 1: Method for assignment of initial arrival times.

```
function InitializePaths(AgentPaths, G(V, E), v)
NumPaths = AgentPaths.Num;
NumFinishedPaths = 0;
while NumFinishedPaths < NumPaths do
    foreach path ∈ AgentPaths do
        foreach vertex ∈ path do
            if vertex.resolved then
                continue;
            AllDependenciesResolved = True;
            LatestDependency = 0;
            foreach dependency ∈ vertex do
                if dependency.resolved then
                    LatestDependency = Max(LatestDependency, dependency);
                else
                    AllDependenciesResolved = False;
                    break;
            if AllDependenciesResolved then
                vertex.resolved = True;
                travelTime = dist(vertex, vertexParent) / v;
                vertex.arrivalTime = Min(travelTime, LatestDependency);
                if vertex.child = ∅ then
                    NumFinishedPaths++;
```

the initial state is not valid.

The initialization works by iterating over all agent paths in an outer loop and iterating over all vertices in a path in an inner loop. In the inner loop, arrival times are assigned as soon as possible with respect to edge lengths, agent constraints and temporal dependencies. When the inner loop arrives at a vertex with a temporal dependency where the time of the preceding event has not yet been determined, the inner loop terminates. Temporal dependencies where the preceding vertex has been assigned an arrival time are referred to as resolved. The outer loop will then continue with the next path. In every complete iteration of the outer loop, some path will advance at least one vertex. When all vertices have been assigned an arrival time, the initiation phase is complete.

3.1.2 Path optimization

The optimization phase iterates through all paths several times, each time attempting to arrive at the goal vertex as fast as possible with respect to dependencies. As a path P_1 is optimized, any path P_2 with dependencies to path P_1 might be able to be optimized further. In some instances two paths might depend on each other, in this case optimization to any of these might allow for further optimization of the other. Because of this, all paths must be continually iterated through and optimized until no changes are made to any paths in a complete iteration of all paths, as shown in Heuristic 2. Next, the optimization process of a single path is explained.

Heuristic 2: Path optimization.

```
InitializePaths(AgentPaths, Agents);
while RemovedSlack > 0 do
    RemovedSlack = 0;
    foreach Path ∈ AgentPaths do
        RemovedSlack += OptimizePath(Path);
```

The goal of the optimization heuristic is to greedily arrive as soon as allowed at every vertex in the path. When optimizing a path every vertex is iterated over, from first to last. The highest allowed acceleration is searched for using a divide and conquer approach, for every vertex.

The values for highest possible acceleration a_{hi}, highest possible deceleration a_{lo}, and currently attempted acceleration a_{cur} are stored. The initial values of a_{hi} and a_{low} are the highest and lowest possible acceleration achievable by the agent. The value of a_{cur} is assigned to be $a_{cur} = (a_{hi} + a_{lo})/2$, it is now assumed that the agent travels over edge $e = \{v_1, v_2\}$ with an acceleration of a_{cur}. The initial velocity is provided by the arrival velocity at v_1. The value of a_{cur} is then evaluated to determine if it is a valid acceleration. There are four conditions that must be met for the acceleration to be considered valid, the first of which is that the arrival speed at v_2 must not be greater than the highest allowed speed for the agent. The second condition is that the arrival time at v_2 respects any temporal dependencies associated with v_2. The third is that if an agent decelerates, the deceleration must

not be so high as to cause the agent to achieve negative velocity, as this would prevent it from ever reaching v_2. Finally, the fourth condition states that the arrival time and speed at v_2 must not force the agent to violate temporal dependencies at vertices following v_2. This occurs when a vertex v_x proceeding v_2 is associated with a temporal dependency that is defined to be earlier than the latest possible arrival arrival time at v_x. In other words, assuming that the agent decelerates as much as it is capable of, it will still arrive earlier than allowed at v_x.

To evaluate whether an agent a can travel to the proceeding vertices after v_2 without violating constraints, it is assumed that a continues travelling with the highest deceleration possible. The arrival time and speed at all proceeding vertices are calculated until either a temporal constraint is violated and hence a_{cur} is too high, or the agent achieves negative velocity over an edge or there are no more vertices on the path. If there are no more vertices on the path or the agent achieves negative velocity over an edge, a_{cur} is considered to be allowed.

If a_{cur} was found to be too high, a_{hi} will be reassigned to be equal to the current value of a_{cur} for the next loop iteration. If a_{cur} was found to be allowed, the value can be used for acceleration when travelling from v_1 to v_2. However, instead of returning at the first found valid acceleration, more iterations can be performed with the goal of finding a better value. If more iterations are desired, a_{low} is assigned to be equal to a_{cur} for the next iteration. In this framework, 30 iterations are always performed and the highest valid acceleration is used.

In Figures 3.1 and 3.2 examples of the heuristic can be shown. In the first iteration, a_{low} denotes the highest possible deceleration and a_{hi} denotes the highest possible acceleration possible for the agent. a_{cur} denotes the acceleration that is evaluated in the current iteration. Figure 3.1 illustrates how each iteration attempts to move a_{cur} towards the span of allowed acceleration values. The first two iterations produce acceleration that is too high, in the third iteration an allowed acceleration is found. Figure 3.2 illustrates how additional iterations can produce a better result. Already in the first iteration an allowed value is found. However, by running two additional iterations a higher allowed value is found.

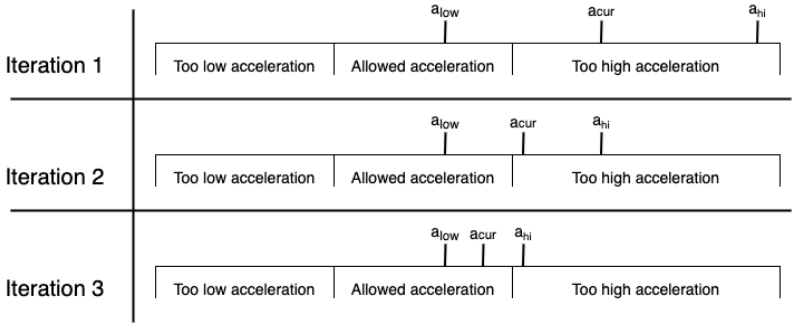

Figure 3.1: Three iterations of optimization where an allowed value is found in the last iteration.

Continuous acceleration is achieved by performing many iterations on the divide and conquer phase. a_{hi} and a_{low} represent a span, the size of this span is halved for every iteration. The goal is to have the span close in around the upper bound of

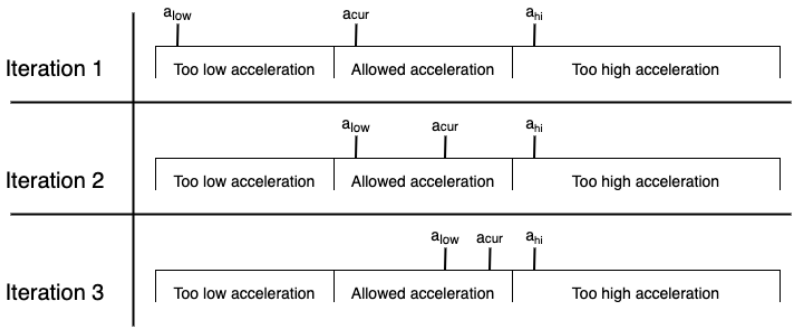

Figure 3.2: Three iterations of optimization where the result gets progressively better.

allowed accelerations. If the original span is 1 second, after 30 iterations, the size of the span will be $1 \div 2^{30}$, which is close to one billionth of a second.

3.1.3 Delay Tolerance

Mobile agents in the target system are sometimes delayed during the execution of a plan. If the distance between agents is very small in the plan, this could cause collisions. One solution to this problem is to stop all mobile agents and recalculate their paths when delays are detected. However, re-planning paths is not an ideal solution as it is time consuming. A better solution is to introduce bounded delay tolerance. If a solution has a delay tolerance of t time units, agents can be delayed up to t time units during the execution phase without risk of collisions.

As previously described, temporal dependencies between agent paths are defined in the TPG structure as Type 2 edges. A Type 2 edge $e = \{v_1, v_2\}$ defines the constraint that the event v_1 must not be scheduled after v_2. The original MAPF-POST algorithm associates these edges with a lower bound of 0 and an unbounded upper bound, which translates to the constraints that v_2 to be scheduled no earlier than v_1 and no later than infinity. Associating a type 2 edge lower bound with a positive value t forces v_2 to be scheduled no earlier than the time of $v_1 + t$. Described in words, v_2 will be scheduled t time units after v_1, this allows the agent that is scheduled to arrive at v_1 to be delayed up to t time units during the execution.

3.1.4 Limitations

The velocity constraints imposed on mobile agents in the target system when turning at high speeds are not considered in this work, the heuristic could be extended to consider this as well. Another assumption is that all agents are disc shaped, as only their radius is assumed. While a disc could be used to approximate more complex shapes, it might impose undesired constraints on the graph.

The goal of the optimization is to arrive at every vertex as soon as possible. This is not always the best choice, as it does not consider arrival *speed*. Sometimes, arriving at a later time with a higher speed is preferable, as it will cause the agent to arrive at succeeding vertices at an earlier time. Extending the heuristic to also maximize arrival speeds might make it produce solutions with lower costs.

Like MAPF-POST, connected vertices must be separated by a certain minimal distance for the result to be guaranteed to not contain collisions[10]. If the distance between two connected vertices is too small, safety marker vertices can not be placed correctly and hence the paths are no longer guaranteed to be free of collisions.

Like MAPF-POST, this heuristic can not prevent collisions where edges cross without a vertex. As crossing edges is a desired feature of the target system, this was a problem that had to be resolved. Edge crossings are removed from the graph by inserting vertices at every crossing in the graph, removing the edge crossings. The crossings are removed by performing intersection testing on all edges against all other edges. A vertex is inserted at the point of intersection, when there is one. The original edges are rerouted to be directed at the inserted vertex. Two edges are added to the inserted vertex and these are directed to the targets of the original edges, as seen in Figure 3.3.

The insertion of extra vertices added new undesirable routes for agents. To prevent agents from using the new paths, a conditional grandparent variable is added to the new edges created when inserting vertices. The conditional grandparent defines a vertex which an agent must have travelled through in order to traverse that edge. This value is used in the low level phase of the abstract models. This can be seen in Figure 3.3

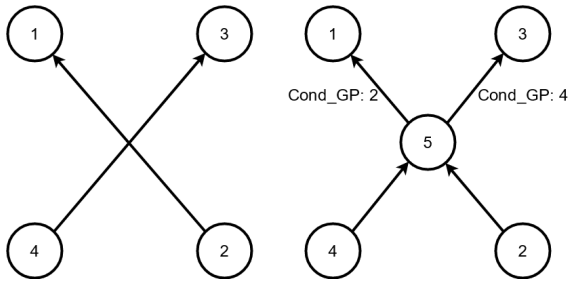

Figure 3.3: Graph after crossing edges have been removed.

3.1.5 Mitigation of limitations

Some effort was put into finding a way to remove the edge distance constraints imposed by the MAPF-POST algorithm. No solution was found, but a a method that could eventually be developed into a solution was found.

The reason that short edges are not allowed in the graph is that in MAPF-POST (which our heuristic is based on), collisions are avoided using temporal dependencies. These are only detected when agents travel through the same vertex, as described in [10]. Consider an example where agent x and agent y travel through the same vertex V, agent x first. The temporal dependency between x and y is not enforced at the vertex V, it is enforced at the generated safety markers, as seen in Figure 2.3 (the dependencies between agents are shown as arrows between the horizontal paths). The safety markers define two positions, one at which y can be located while travelling towards V, and one that defines a position at which x can be located while leaving V, at the same point in time, without collisions. Every edge $e = \{v_1, v_2\}$ in a graph must be long enough for both a safety marker that defines a location at a

safe distance to v_1 and a safety marker that defines a location at safe distance to v_2. If the edge is too short, both safety markers will not fit on the edge.

As the Type 2 edges do not represent a path, or anything else than a temporal dependency, our hypothesis is that they can be redirected. This would cause the temporal dependency to be moved to an event that is earlier in the target agents path. An example of this proposed procedure can be seen in Figure 3.4. In this example, the red edge in a) is found to be too short and might cause a collision. However, the physical distance is increased by rerouting the target vertex to one that is earlier in the path of agent 2, as seen in b). Another approach is to do the opposite, move the source of the edge forward in time instead of moving the target of the edge backwards in time. While this proposition is promising, the side-effects of rerouting edges in this fashion is not udnerstood.

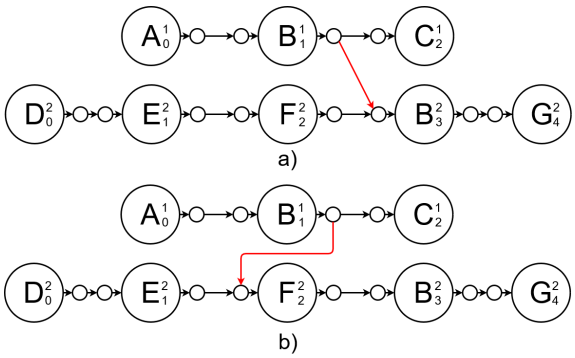

Figure 3.4: TPG graph with rerouted Type 2 edges.

3.2 High level search method

During the search in the high-level constraint tree, all models must always open the node with the lowest cost in order to maintain optimality. This essentially makes the high level search a breadth first search, as new high-level nodes are at least as expensive as their parent. Modifying the high level search to always open the last added node makes the algorithm search on the depth first. This will make the algorithm sub-optimal in the abstract model but it might reduce the time required to find a solution. A demonstration of the expected behaviour is shown in Figure 3.5. Here, we can see how the constraint tree is expanded from left to right.

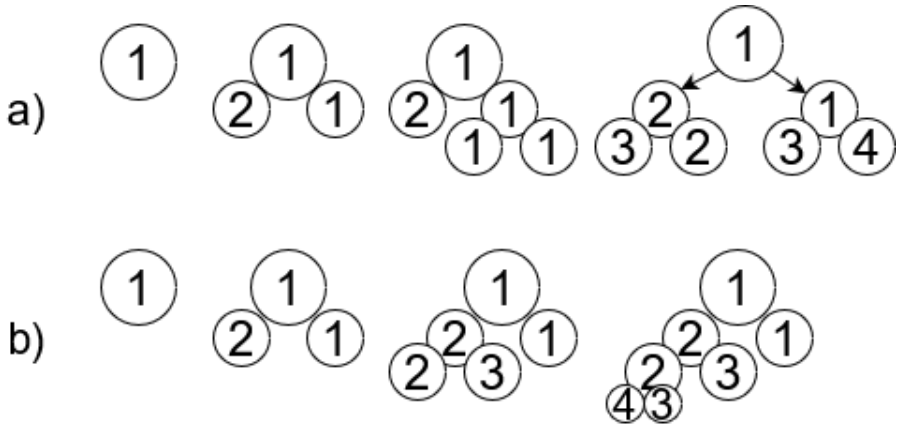

Figure 3.5: Mockup constraint trees generated using BFS (a) and BFS (b) search. Costs of the nodes are shown in the circles.

3.2.1 Method for parallel high-level search

A method for searching in the high-level constraint tree in a parallel manner was developed. As the solvers of all models perform this search in a similar fashion, the method is applicable to every considered model of this work. The method works by having several workers which all fetch open nodes from the same constraint tree. A worker is some hardware or software capable of solving low level nodes, for example a whole system or threads on the same CPU. As described in Section 2.1, once a node has been fetched from the constraint tree the node is solved in the low-level. Some synchronization primitive must be used to ensure that threads do not interact with the constraint tree simultaneously. Interaction is the act of fetching a new open node or adding a new node to the tree. This is mainly to prevent workers from attempting to solve the same node. Also, depending on what structure is used to represent the constraint tree, simultaneous interaction with the constraint tree might cause it to enter an invalid state. The workflow is presented in Figure 3.6. When a solution is found, access to the constraint tree is blocked and workers are stopped.

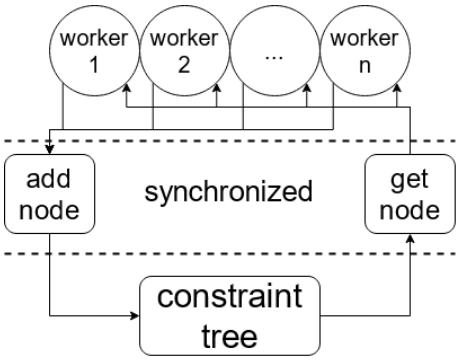

Figure 3.6: Parallel solver workflow.

4

Framework

This chapter contains a description of the workflow and features of the developed framework.

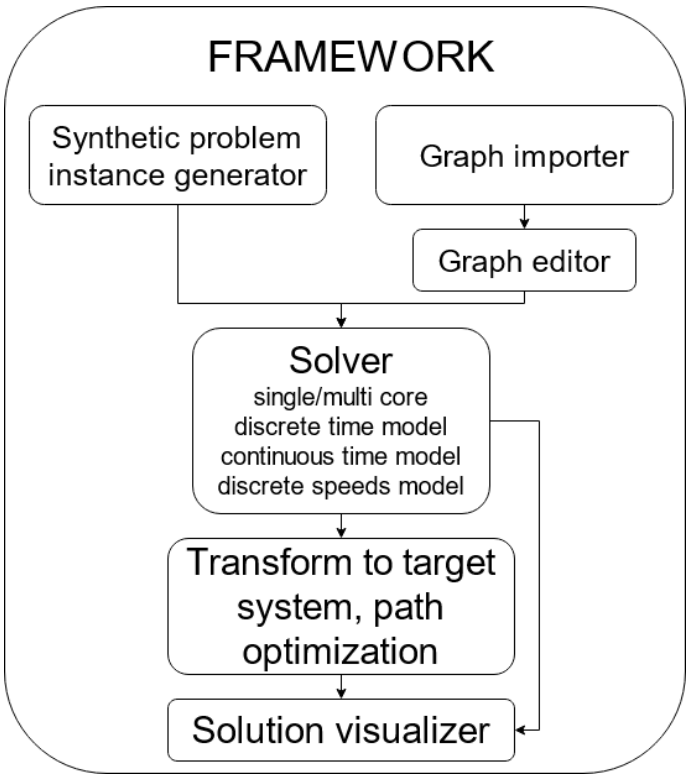

Figure 4.1: Flow and components of our framework.

4.1 Framework structure

The general workflow of the framework created in this work can be described as follows. An input graph is provided by the graph importer or one of the problem instance generators. Crossing edges in the graph are removed algorithmically. The solver is invoked with the problem description and solver parameters. The solver parameters consists of which model to use, which solver to use and the search configuration to be used. The problem description consists of the graph and agent data. The agent data consists of agent count, start- and end positions, top speeds and sizes. If a solution is successfully found it is transformed to the target system using

the enhanced MAPF POST heuristic. The resulting solution is exported to a file that can be used to visualize the solution. This workflow is shown in Figure 4.1

4.2 Framework features

4.2.1 Visualizer

Visualization of graphs and problem solutions is useful for debugging and inspecting interesting cases. A visualiser was created using Unreal Engine 4, it uses data produced by the solver to visualize agent solutions and graphs. For the discrete time and continuous time models, agents are interpolated along their paths in constant speed. For the discrete speeds model and the target system, agent acceleration is also incorporated.

4.2.2 Graph editor

To the end of making it easier to manually create test cases, graph editor was created and incorporated to the visualizer. Manual cleanup and visual inspection of imported roads is possible, but new graphs can also be created and exported to the solver.

4.2.3 Automated generation of graphs using publicly available maps

Sections of publicly available road data can be imported to the editor using OpenStreetMaps.com. This data contains redundant information such as houses, foilage, walking streets. The redundant data is removed with a parser which also converts the .osm file exported from https://www.openstreetmap.org/ into a .csv file which is used in the engine. The parsing is done by inspecting the .osm file and isolating the structures of the roads. Once the roads are identified everything else is removed and the road structure of the map is written to a .csv file. Imported graphs must also be examined to ensure that lanes are running in the correct direction, no parts of the graph are inaccessible and roads have lanes in both directions. Typically these conditions are not fulfilled and extensive editing of the graph must be performed manually before the graph can be used.

Figure 4.2: Generated graph as shown in the editor and the original map, shown in OpenStreetMaps.com.

5
Evaluation Plan

With the intention of evaluating the proposed models and our own contributions, we propose an experimental evaluation of the models and features of the framework. The focus of the evaluation is to answer the research questions defined in Section 5.1. In this chapter the planned experiments are presented and the evaluation environment is described.

The experiments showed that the continuous time model generally has the highest success rate. We have also observed that the overhead introduced by transforming solutions to the target system is several orders of magnitude smaller than the time required to produce solutions.

5.1 Research questions

The studied problem has detailed representation of the target system (TS). Gerdin [9] studies trade-offs between the granularity of these details and the computational costs. The less granular model considers a discrete-time model with uniform speeds, this is called the discrete-time model (DT). The most granular model consider continuous-time and a discrete set of speeds, this is called the continue time model with discrete speeds (CTDS). Between these models lay the continuous-time model (CT), which considers uniform speeds. The proposed framework aims at balancing this trade-off by converting an instance of the studied problem from its representation for the target system, I_{TS}, to its representation for an abstract model, $C_{TS \to x}(I_{TS}) : x \in \{DT, CT, CTDS\}$. The question is how to perform this conversion efficiently.

Another set of questions is related to the efficient implementation of algorithms that solve the MAPF problem in each of these models. We are interested both in sequential solutions as well as solutions that harvest the power of multi-core CPUs. We note that this set of questions is significant since its complexity is known to be NP-hard [19] and thus an inefficient implementation of the framework can render its performance to be impractical.

As presented by Sadikov et al.[16], it is possible to speed up the single agent path finding algorithm A-star by using heuristics that are not admissible, called pessimistic heuristics. We note that A-star used with a pessimistic heuristic is suboptimal. As the abstract models considered in this work all use some variation of A-star to solve single agent path finding problems, it is possible to apply pessimistic heuristics to all of them. In a similar way, CBS is only guaranteed to be optimal if the high level search always opens the node with the lowest cost, essentially

performing a breadth first search. We are interested not only in how the target system performance is affected by the use of suboptimal solvers, but also how this affects the success rate and runtime of the abstract models.

It is worth mentioning that no solution produced for any abstract model has guarantees to be valid for mobile agents in the target system. Thus, once the solution, $R(I_x)$, in the abstract model $x \in \{DT, CT, CTDS\}$ is available for an instance of the MAPF problem, I_x, there is a need to efficiently transfer this solution to one that can serve as a valid solution for the target system. We denote this transformation by $T_{x \to TS}()$ and write $T_{x \to TS}(I_x)$ when we refer to the solution obtained for the target system. Such transformations have also to take into account the efficiency of the transferred solution since its guarantees for optimality, which held for the abstract model, no longer exist.

Perhaps the most important set of research questions of this project are related to the relation between the above abstract models when the framework performs a complete circle, see Figure 5.1. That is, for which $x \in \{DT, CT, CTDS\}$ does $T_{x \to TS}(C_{TS \to x}(I_{TS}))$ have attractive properties that are related to our evaluation criteria as well as the studied test cases. We expect the answers to these questions to inform which abstract model is the most worthwhile to use when considering different applications.

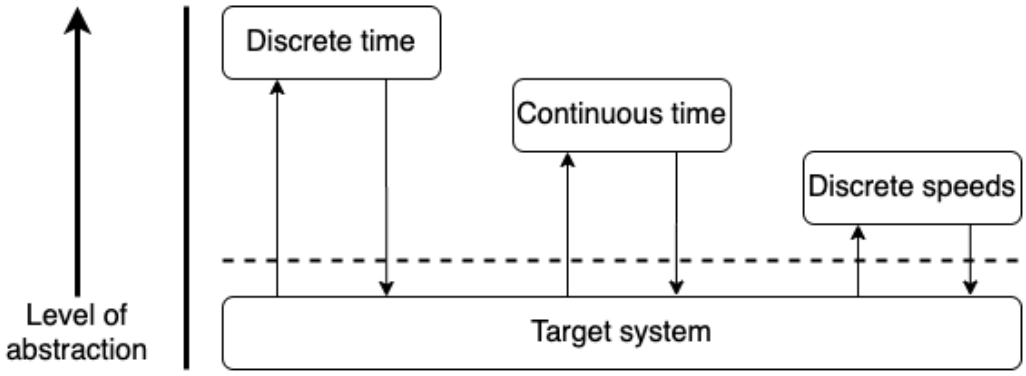

Figure 5.1: Transformation flow between abstract models and the target system.

5.2 Evaluation criteria

To evaluate the proposed models, optimization methods and framework implementation, several sets of experiments are performed. These will be directly related to the set of research questions that define the areas of interest for this work. A set of criteria is collected during the execution of all experiments, these are used to quantify performance.

The time required to solve a problems is measured, this time is denoted *runtime*. A problem being solved in less than 5 minutes is considered a success, if more than 5 minutes is required it is considered a fail. The success rate of all experiments is measured by running the experiments using 10 problem instances with randomly generated agent data. Efficiency of the produced solutions is measured by calculating

the SIC and makespan before and after transformation to the target system. The efficiency of transformation between the abstract models and the target system is quantified by the time it takes to perform the transformation. The number of explored high-level nodes opened for each problem is also considered, this can be used to study algorithmic behaviour but it is also a useful performance metric.

5.3 Use cases and test cases

Our evaluation plan considers graphs that were generated either from publicly available maps or synthetic data. The former represents the studied use case whereas the latter allows us to evaluate the proposed framework using smaller interesting cases.

5.3.1 Use cases

The use case performance will be evaluated on two graphs generated from publicly available maps. One from an urban area and one from a sparsely populated area. An example map can be seen in Figure 4.2. These cases are interesting as one of the goals of this work is to make MAPF applicable to mobile agents in transport systems. However, these graphs will not be the center of focus during evaluation as a more efficient evaluation can be performed using synthetic graphs.

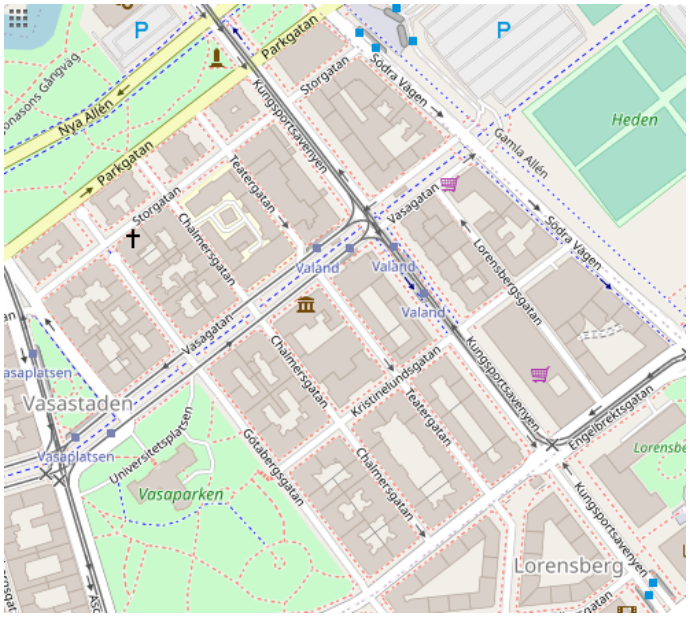

Figure 5.2: Map used as dense map in the use cases.

Figure 5.3: Map used as sparse graph in the use cases.

5.3.2 Test cases

Test cases is the main focus of the evaluation in this work. The isolated scenarios are thought to be more prune to collisions. While these graphs are not generated from publicly available maps, they are designed to represent interesting scenarios from road networks. The considered test cases can be seen in Figures 5.4, 5.5, 5.6.

Problem instances are generated dynamically with randomly generated agent start and goal locations. The number of agents in a problem instance is chosen with a parameter to the problem generator. The framework can generate four different synthetic graphs; highway Figure 5.4 , intersection Figure 5.5, roundabout Figure 5.6 and grid Figure 5.7. The grid and highway graph can also be generated to be arbitrarily wide and long.

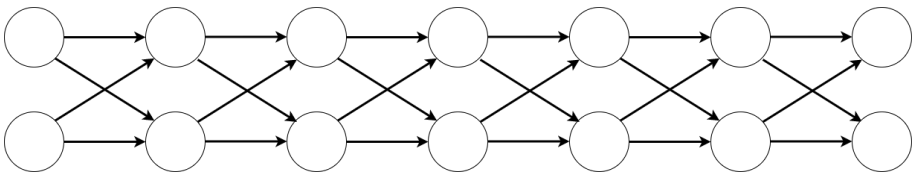

Figure 5.4: Structure of a generated highway used in the experiments.

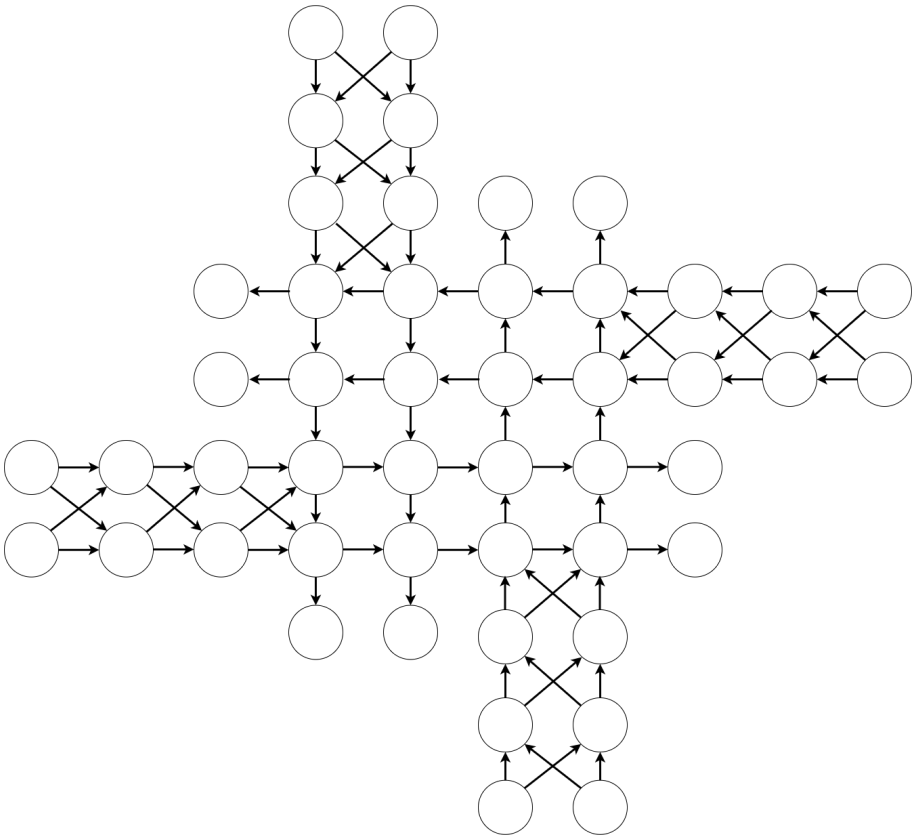

Figure 5.5: Structure of a generated intersection used in the experiments.

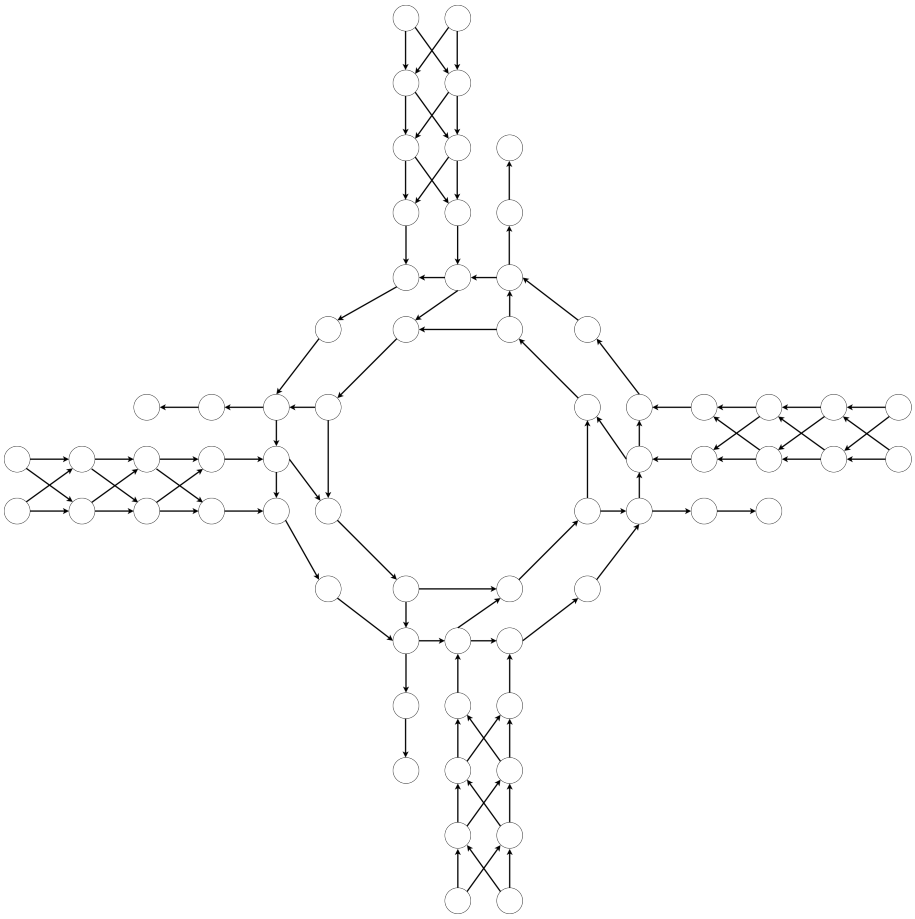

Figure 5.6: Structure of a generated roundabout used in the experiments.

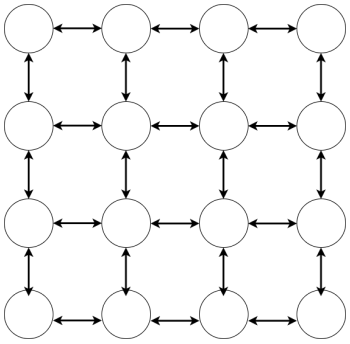

Figure 5.7: Structure of a generated example 4x4 grid.

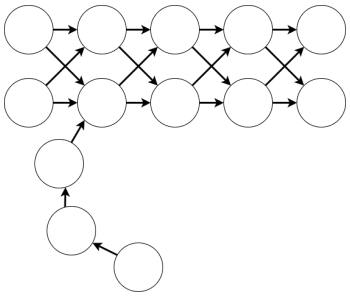

Figure 5.8: Structure of a generated highway entry.

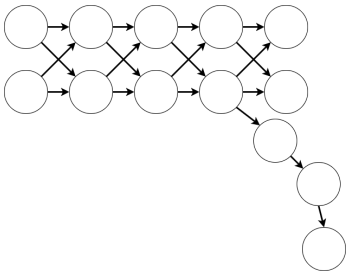

Figure 5.9: Structure of a generated highway exit.

5.4 Evaluation Environment

One system is used to perform experiments, it is equipped with a quad core Intel Core i5-4690k CPU with 16GB ram.

It is assumed that agents leave the graph as soon as they reach their destinations and thus cannot cause collisions after that. This is to simulate a limited section of real roads where vehicles actually continue travelling after reaching their destination.

It is assumed that agents simulated in the CTDS model can always select whatever speed they find best as target speed. This is to not impose additional constraints on this specific model compared to the other models. All experiments use three speeds for the CTDS model, unless stated otherwise.

5.4.1 Problem generation

As previously mentioned, all experiments are performed using 10 different problem instances, since this enables us to observe what the typical result looks like. The problem instances in our framework are defined by a graph, agents and a solver configuration, which includes which model to use and which search methods to use. Typically, the only varying factor in experiments are the agents, which not only vary in number, but also their start and goal vertices. The number of agents is always defined per experiment description. The start and goal vertices are generated randomly to enable us to perform many experiments. One problem with randomly generated problem instances is that there is a risk that the results are skewed. For example when comparing model performance, the problem instances generated for some model might be harder to solve than the generated problems for some other

model. To prevent this, the same 10 seeds are always used to generate agent start positions and goal positions. In the following sections, the method for agent data generation is explained.

The set of possible agent start and goal vertices are defined as follows. For each graph, a pool of possible start and goal vertices are chosen manually to ensure that all goal vertices can be reached from all start vertices. The possible start vertices are defined to be vertices that are close to the first vertices on a road. We chose not to enable agents to start in the middle of a road, as this would likely often cause them to not interact with other agents and thus would not provide any valuable data. The possible goal vertices are defined to be the vertices without outgoing edges, for example the end of a road.

The specific agent start and goal vertices for each problem is generated as follows. Consider an example experiment where four agents are required. The first step is to pick the first four elements from the set of possible start vertices. This step is performed to ensure that the generated start vertices are close to each other, increasing the probability of agent interactions. The second step is to shuffle this array to randomize the order of elements. This step is performed by swapping the location of two randomly selected elements in the set of possible start locations. The element swapping procedure is executed 10 times the number of agents, 40 times in our example case. After the set has been randomized, agents receive a start location from the set. This is done by matching element indexes in the start vertex set to agent indexes in the agent set. Agent goals are found by generating a random number that is no larger than the number of goal vertices, retrieving the goal vertex with the random number of index in the set of goal vertices, and assigning that to the agent. The method for finding agent start vertices is more intricate than the method for finding goal vertices since it is acceptable for agents to have the same goal vertex, but no the same start vertex.

5.5 Experiments

This section explains how experiments are structured to evaluate our research questions. In all the experiments a multi-core implementation of the solver is used unless stated otherwise.

5.5.1 Efficient conversion from target system to abstract model

Our evaluation of this step focuses on efficiency with respect to the processing time required to convert a map section from OpenStreetMaps.com to the structure used by the abstract models. The experiment is conducted by considering five different map sections with sizes ranging from 1 to 5 square kilometers.

5.5.2 Efficient implementation of the framework

The evaluation of the optimal multi-core implementation is performed by comparing the success rate and runtime with that of a single core implementation. These

experiments are conducted using the intersection graph and the three considered models with a varying number of agents, ranging from 2 to 15.

5.5.3 Problem representation in the discrete speeds model

In the case of the discrete speeds model, additional details of the target system can be introduced. By increasing the number of speeds that agents can travel with, the agent representation in the abstract model is more accurate. Yet another way to lower the level of abstraction is to introduce several different options that mobile agents at the target system have when switching lanes on a highway. In these experiments we are looking at the runtime required to solve the problems and the costs of the solutions.

1. The grid graph is used with a varying number of agents, ranging from 1 to 10, and varying numbers of possible speeds, ranging from 2 to 5.
2. The highway graph is used with a varying number of agents, varying from 2 to 10, and with varying number of possible lane switches, ranging from 1 to 3

5.5.4 Performance of suboptimal search methods

Using suboptimal search configurations might negatively affect solutions produced at the abstract models, but it is not known how the runtime or target system solutions are affected. We are interested in how these effects translate to the target system and how it affects computational costs.

1. Low level heuristics is evaluated using the intersection graph with admissible, $2 \cdot$admissible (pessimistic) and $4 \cdot$admissible (more pessimistic) heuristics. All three models are considered and a varying number of agents, ranging from 2 to 15.
2. Using the intersection graph, the suboptimal high level search method, depth first, is evaluated considering the three abstract models and a varying number of agents, ranging from 2 to 15.

5.5.5 Efficiency of transformation from abstract model to target system

The runtime of the transformation is important. If the transformation introduces a significant overhead, it is not worthwhile to transform solutions as a post processing step. The preferred option would then be to develop new models that are closer to the target system. The efficiency of this process is evaluated by measuring the time required to transform solutions from the abstract models to the target system.

5.5.6 Efficiency of abstract models at target system

The selection of abstract model affects the solver runtime and solution costs at target system. Here, we evaluate how the choice of abstract model affects the evaluation criteria in different scenarios. Target system efficiency is evaluated using each abstract model, considering all test- and use case graphs with a varying number of

agents, *i.e.*, 2 to 15. The evaluation criteria we are interested in is the runtime and solution costs at target system.

6
Results

In this chapter results will be shown with explanation for each result. Many graphs are visibly missing data, this is an indication of a failure to solve the problem.

6.1 Efficient conversion from target system to abstract model

The results of the first experiment, considering the runtime of the conversion from the target system to an abstract model, is Presented in Figure 6.1. As expected, the time required to process a map scales with the size of the map. However, we did not expect that the processing time would grow this aggressively. A far larger issue than the processing time is the quality of the result. Roads are typically not oriented in the correct direction, some intersections do not connect fully, two-way roads are often represented as a single lane road and roads unavailable to cars, such as walkways, are sometimes included. To use the data directly without manual inspection and editing would require a more complex conversion method or a better source of data.

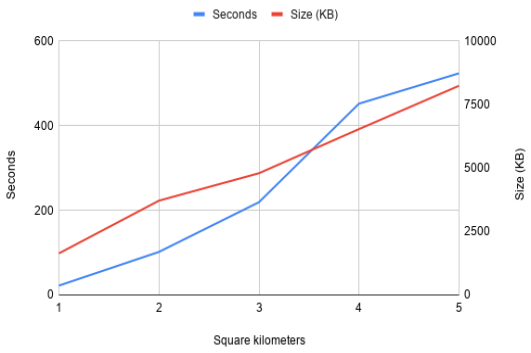

Figure 6.1: Transformation time, target system to abstract model

6.2 Efficient implementation of the framework

In this experiment the multi-core solver was evaluated, the measured criteria is the runtime required to solve problems using the different abstract models. The CPU

that this experiment was performed on has four cores and not simultaneous multi threading.

The runtime result is shown in Table 6.1. The numbers represent the normalized runtime of the multi-core implementation, relative to the serial implementation. In other words, the runtime was reduced with the multi-core implementation when the relative time is smaller than 1.

We observe that the DT model does not see a reduction in runtime from using the multi-core version. The CT and CTDS models see a reduction in runtime for larger numbers of agents.

The increased runtime when using the multi-core implementation for low numbers of agents are expected, as very few collisions are found. Few collisions translates to few nodes in the constraint tree, which means that most of the workers are idle. Overhead from using synchronization is still added, causing the runtime to increase.

As stated above, we observe that results differ between models. Our hypothesis as to why, is as follows. When the abstract model has more details of the target system, the low level runtime increases. The difference between the serial and the multi-core versions of the models is that in the multi-core version there are more workers using the constraint tree as a shared resource. In our implementation, the worker threads need access to the shared resource, the constraint tree, every time they need to get a new node or add a new node. As the computational complexity of the low level decreases, the overhead of having several worker threads increases in proportion and hence the difference in runtime between the multi-core and serial implementation is reduced. Additionally, the graph used for this experiment is small, causing the runtime of each low-level iteration to be low, further compounding the issue.

Agents	DT	CT	CTDS
2	1.28	1.06	1.31
3	1.2	1.03	1.32
4	1.26	0.92	0.88
5	1.18	0.9	0.89
6	1.31	0.97	0.82
7	1.49	0.92	0.81
8	1.64	1.16	0.8
9	1.03	0.84	0.8
10	0.73	6.16	1.24
11	1.1	0.87	0.781

Table 6.1: Parallel solver results.

6.3 Problem representation in the discrete speeds model

In this section we present the effects of having different numbers of possible speeds in the CTDS model, four different sets of speeds are used in these experiments. The effect of having additional lane switch edges on the highway graph is also evaluated

in this section.

6.3.1 Grid graph with varying number of speeds

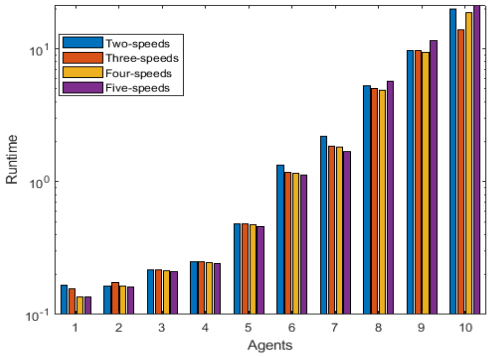

Figure 6.2: Grid runtime

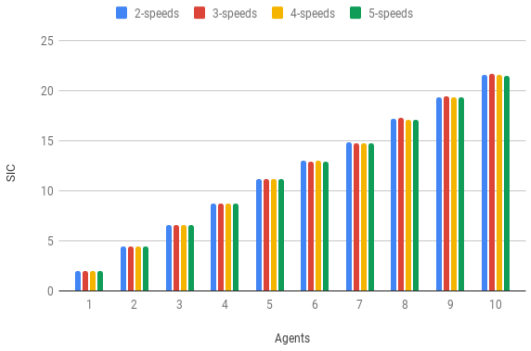

Figure 6.3: Grid SIC

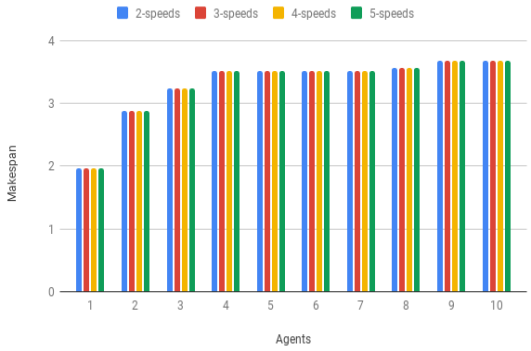

Figure 6.4: Grid makespan

In Figures 6.3, 6.4 and 6.2 we present the SIC, makespan and runtime when evaluating multiple speeds on the grid graph. There is a negligible difference in SIC and makespan. This is expected as the introduction of more speeds does not enable agents to arrive at the target faster. The runtime is not significantly affected by the number of speeds.

6.3.2 Intersection graph with varying number of speeds

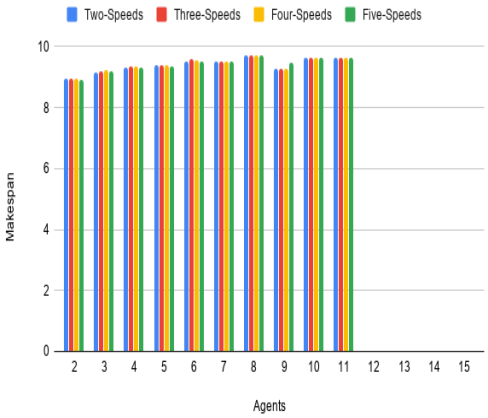

Figure 6.5: Intersection Makespan

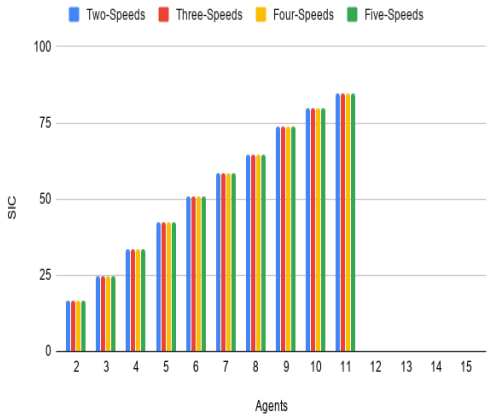

Figure 6.6: Intersection SIC

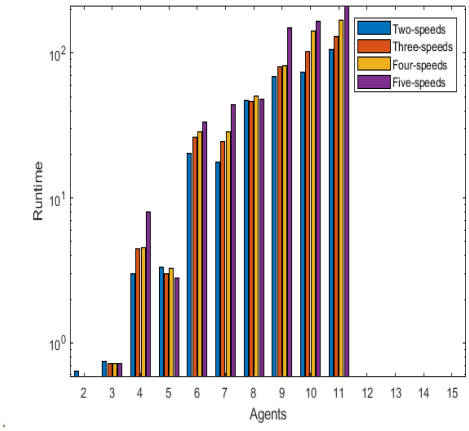

Figure 6.7: Intersection Runtime

On the intersection graph the makespan and SIC does not differ significantly between speed sets, as seen in Figures 6.6 and 6.5. The runtime, seen in Figure 6.7, tends to be lower when smaller sets of speeds are used.

6.3.3 Roundabout graph with varying number of speeds

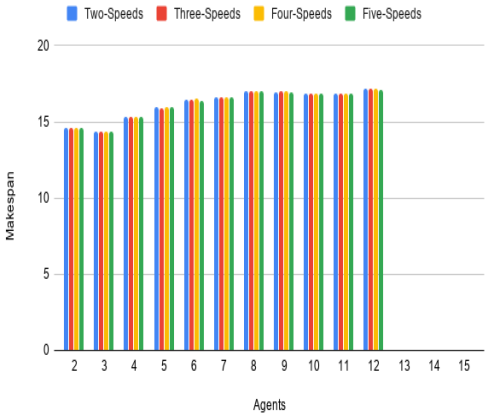

Figure 6.8: Roundabout Makespan

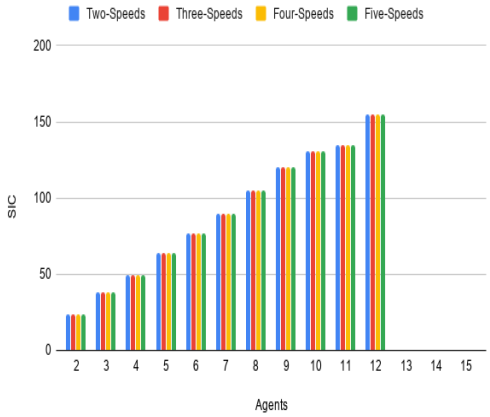

Figure 6.9: Roundabout SIC

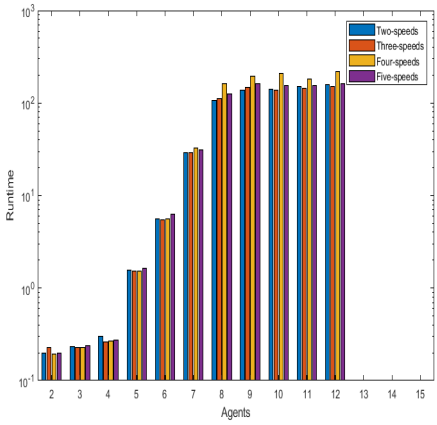

Figure 6.10: Roundabout Runtime

On the roundabout graph, there was a negligible difference in makespan and SIC between the different sets of speeds, as presented in graphs 6.8 and 6.9. In contrast to the other experiments, the runtime is consistently higher when using four speeds than it is when using five speeds, seen in Figure 6.10. However, the difference is small.

6.3.4 Highway graph with varying number of speeds

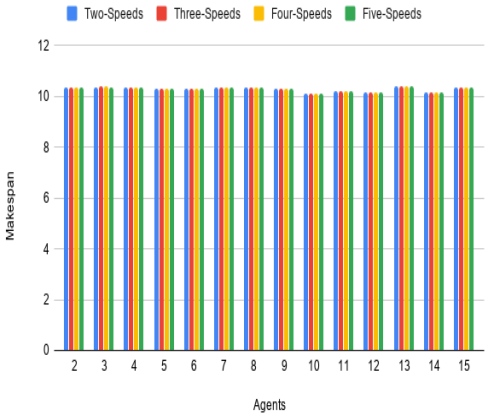

Figure 6.11: Highway Makespan

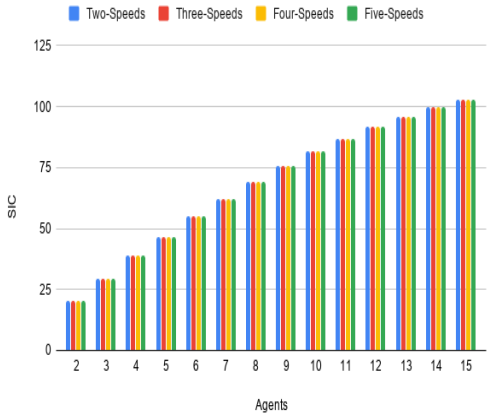

Figure 6.12: Highway SIC

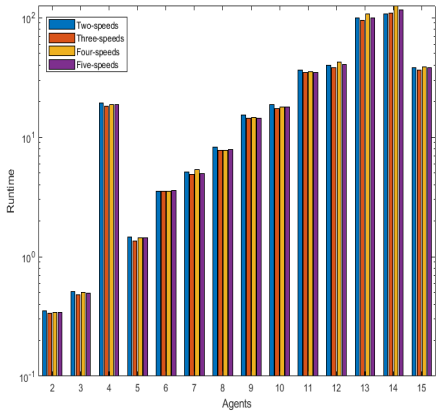

Figure 6.13: Highway Runtime

On the highway graph there was a negligible difference in all measured criteria, presented in Figures 6.11, 6.12 and 6.13.

6.3.5 Highway Exit graph with varying number of speeds

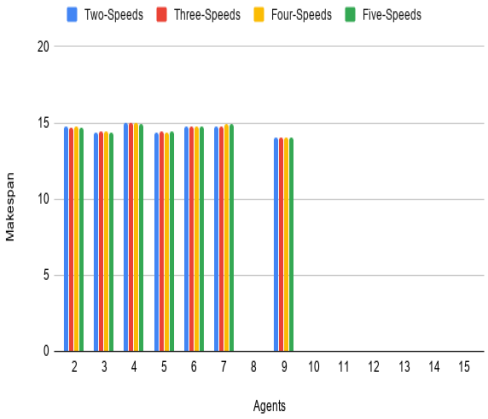

Figure 6.14: Highway Exit Makespan

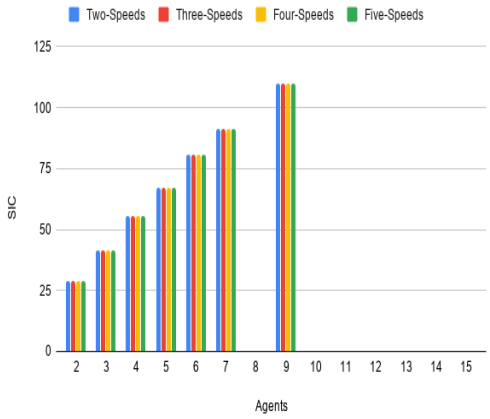

Figure 6.15: Highway Exit SIC

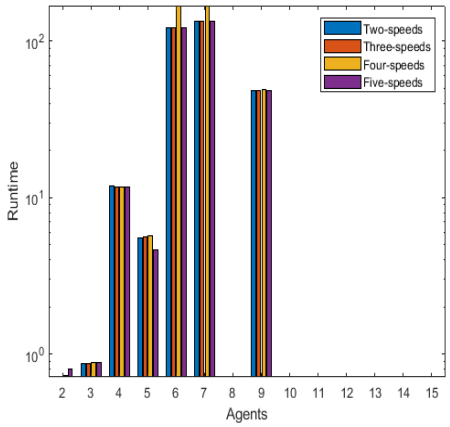

Figure 6.16: Highway Exit Runtime

Here we present the resulting SIC 6.15, makespan 6.14 and runtime 6.16 when evaluating multiple speeds on the highway exit graph. Here we see a volatile runtime when compared to the other experiments, this is a consequence of a high failure rate for this graph. Here we see that the number of available speeds has a negligible effect on costs, but with four speeds it takes more time to find a solution.

6.3.6 Highway Entry graph with varying number of speeds

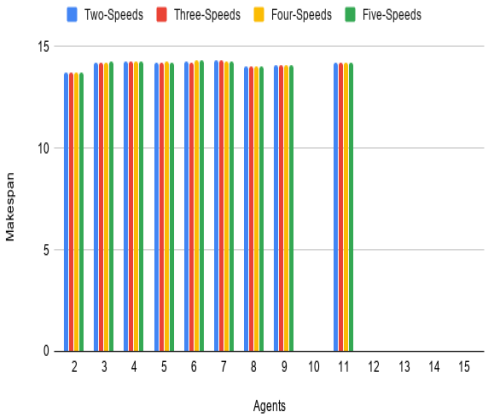

Figure 6.17: Highway Entry Makespan

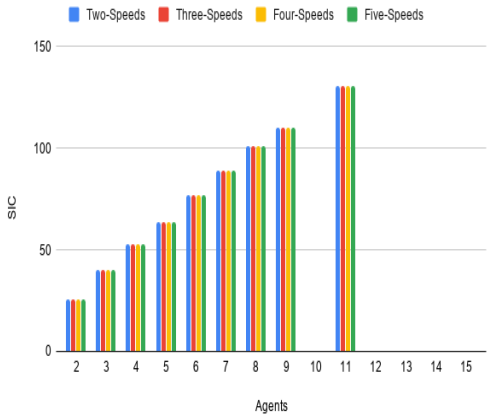

Figure 6.18: Highway Entry SIC

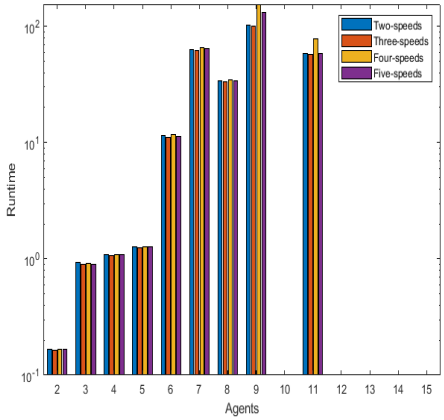

Figure 6.19: Highway Entry Runtime

The SIC, makespan and runtime for the highway entry graph is presented in Figures 6.18, 6.17 and 6.19. Problems on this graph was challenging to solve as no solutions were found for high agent counts. This is probably also the reason why the runtime graph is so volatile and not growing consistently like earlier experiments.

6.3.7 Highway graph with varying number of lane switch edges

We expect an improvement in SIC and makespan since the new lane switch edges introduces a shorter path from the start vertices to the goal vertices, however this reduction in path length is small relative to the total length of the road. As for runtime, it is expected to grow as more lane switch edges are introduced. The increased runtime will also cause the success rate to fall, as the the new edges introduces more crossing edges and hence, more edges for agents to collide on.

As shown in Figures 6.21 and 6.22, increasing the number of available lane change options decreases the makespan and SIC slightly. The runtime increased with the number of possible lane changes, seen in Figure 6.20. We observe that the runtime for the tests using no skips and single skip has a similar runtime, with the no skip results having the lowest.The success rate of this experiment is presented in Figure 6.23, see that the double skip has a low success rate compared to the single skip and no skip results, as these don't fail on a single problem instance.

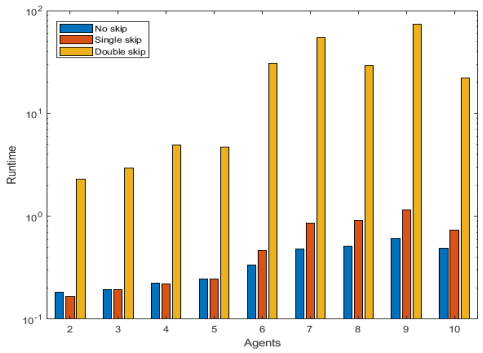

Figure 6.20: Highway with different numbers of skips, runtime.

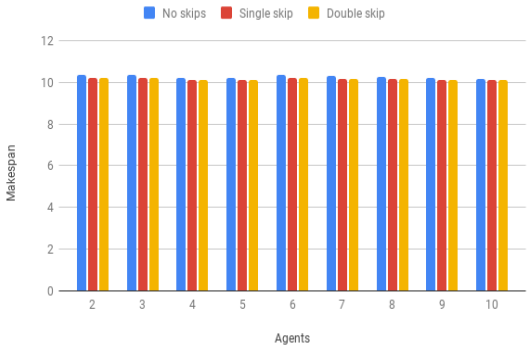

Figure 6.21: Highway with different numbers of skips, makespan.

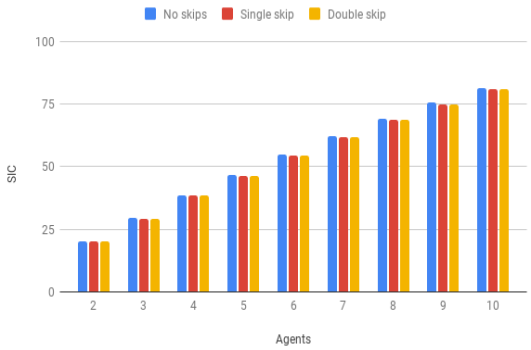

Figure 6.22: Highway with different numbers of skips, SIC.

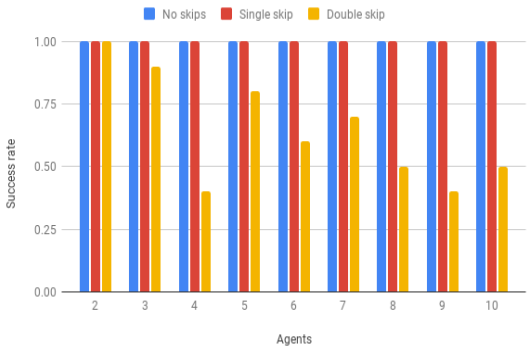

Figure 6.23: Highway with different numbers of skips, success rate.

6.4 Performance of suboptimal search methods

6.4.1 DFS and BFS search

For this experiment we expect the runtime to be reduced when DFS is used as collisions are resolved faster, but in an suboptimal manner. As a consequence, the makespan and SIC are expected to be larger when using DFS.

The measured runtime when using high level DFS search for the different models is presented in Figure 6.24 and the BFS runtime is presented in Figure 6.25. We observe that runtime has been reduced for all models when using the DFS method.

This in turn has caused the success rate to increase, as seen by the absence of values in the BFS graph when higher numbers of agents are being used.

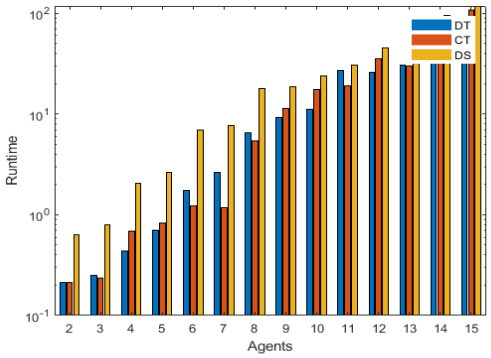

Figure 6.24: DFS runtime

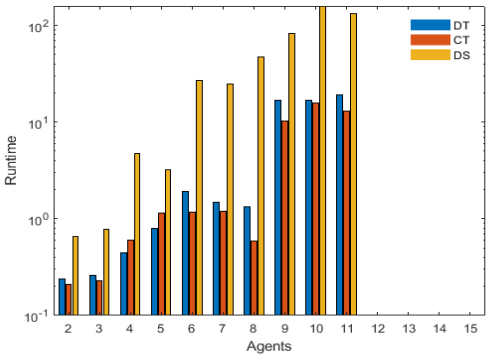

Figure 6.25: BFS runtime

Contrary to expectation, the increase in makespan, seen in Figures 6.28 and 6.29, and SIC, seen in Figures 6.26 and 6.27, is small when compared to the improvements in runtime. This applies to all models and all ranges of agents. While the reason for this behaviour is unknown, one hypothesis is that in this type of graph the way conflicts are resolved is not important. Individual conflicts are still being resolved in an efficient manner as the constraint tree is still being used in the same way, but different nodes are being opened in the high level.

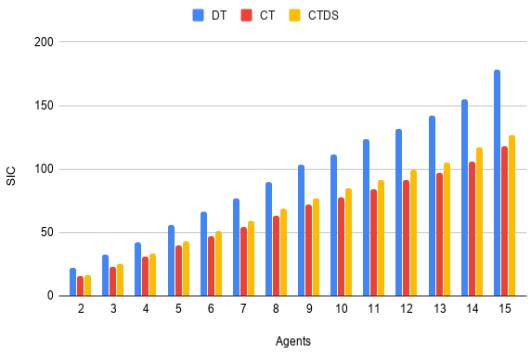

Figure 6.26: DFS SIC

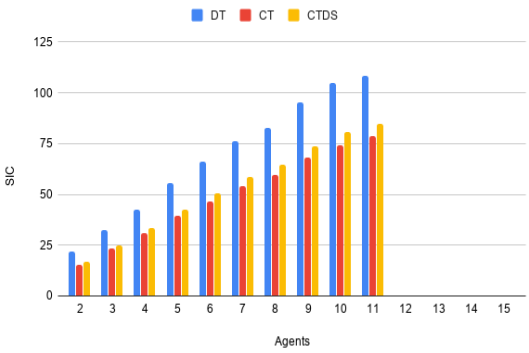

Figure 6.27: BFS SIC

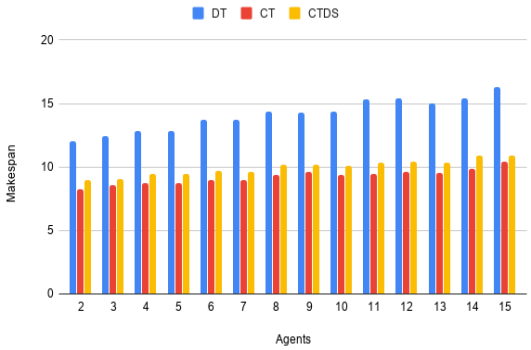

Figure 6.28: DFS Makespan

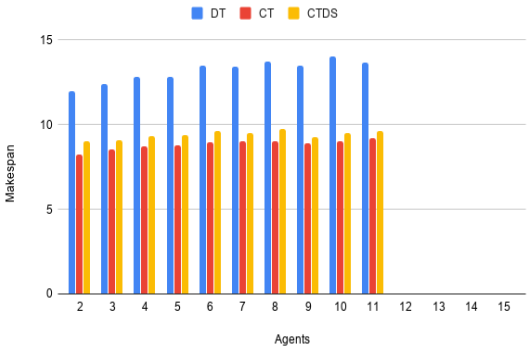

Figure 6.29: BFS Makespan

6.4.2 Admissible and pessimistic heuristics

In this experiment we expect the runtime to be lower when pessimistic heuristics are being used, as the computational requirements of the low level search should be reduced. However, the SIC and makespan are expected to be lower for the admissible heuristic as it produces optimal solutions. The costs presented are not transformed, meaning that the makespan and SIC should not be compared between models.

The runtime results for the DT, CT and CTDS models can be seen in Figures 6.30, 6.32 and 6.31 respectively. In the CTDS model we observe the expected behaviour with pessimistic methods generally having a lower runtime. However, In the DT and CT models there is no apparent pattern as to which model produces the lowest or highest runtime. The reason behind these results are unknown, more experiments are required to understand them.

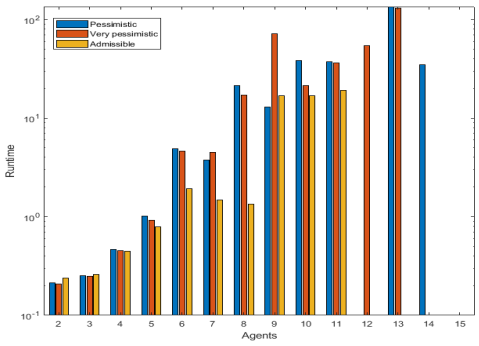

Figure 6.30: Discrete time runtime

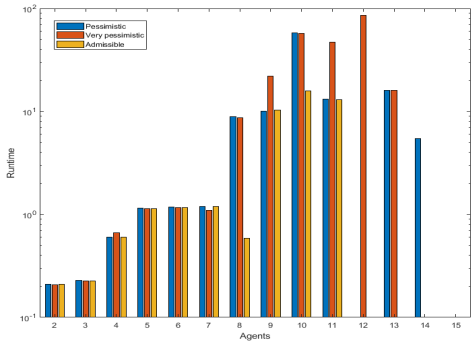

Figure 6.31: Continuous time runtime

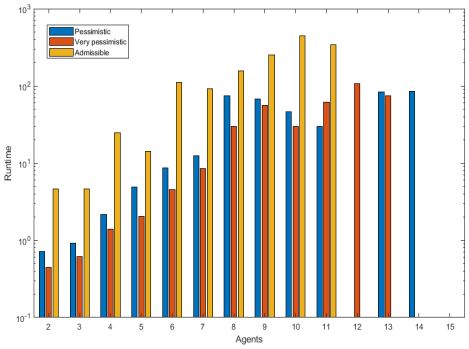

Figure 6.32: Discrete speeds runtime

The makespan results for the different search configurations are presented in Figures 6.33, 6.34 and 6.35. As expected, pessimistic heuristics result in a higher cost when compared to admissible.

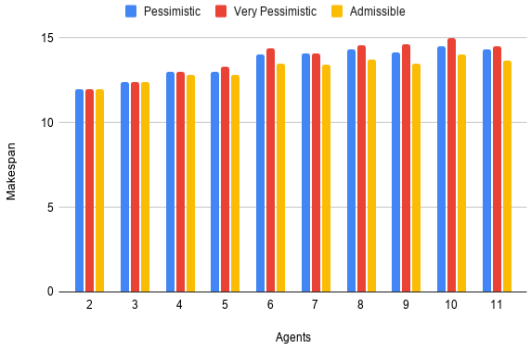

Figure 6.33: DT

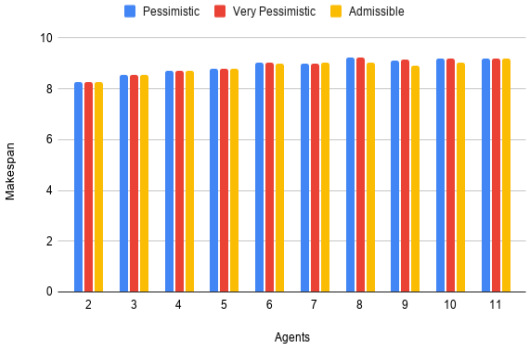

Figure 6.34: CT

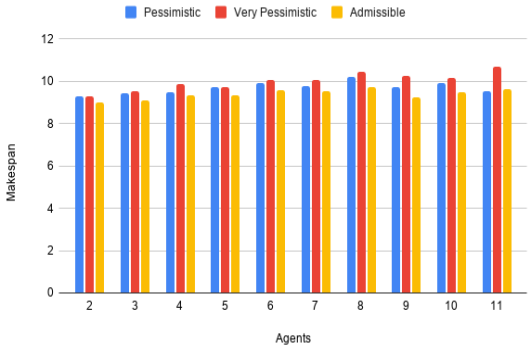

Figure 6.35: CTDS

The SIC results for the different search configurations are presented in Figures 6.36, 6.37 and 6.38. Again we observe that the methods produce similar costs.

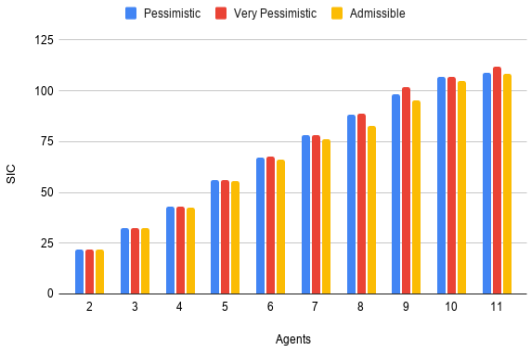

Figure 6.36: DT

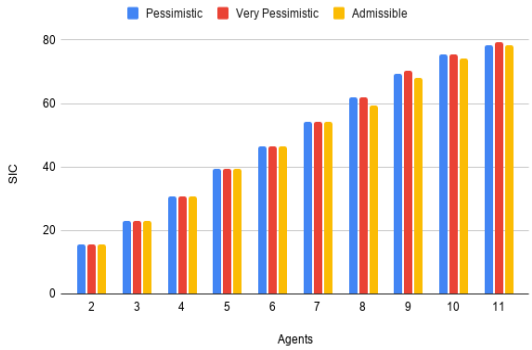

Figure 6.37: CT

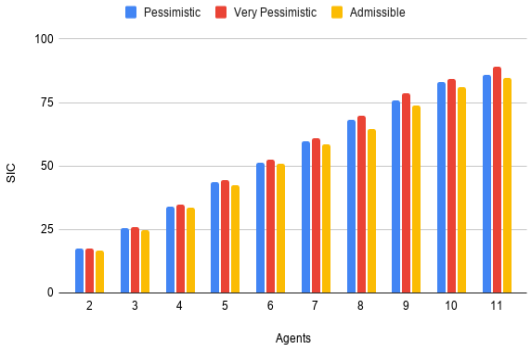

Figure 6.38: CTDS

6.4.3 Performance of suboptimal methods in the target system

Our expectation for this experiment is that the costs at the target system trends to being higher for all models when using suboptimal methods. However, this is unlikely to always be the case as the properties of the agents change when transformed to the target system. We expect the difference in cost between optimal and suboptimal to be small, as the paths are optimized in the target system. The cost difference is expected to be larger in the heuristics experiment than the DFS experiment as pessimistic heuristics has a risk of producing suboptimal paths in the low level.

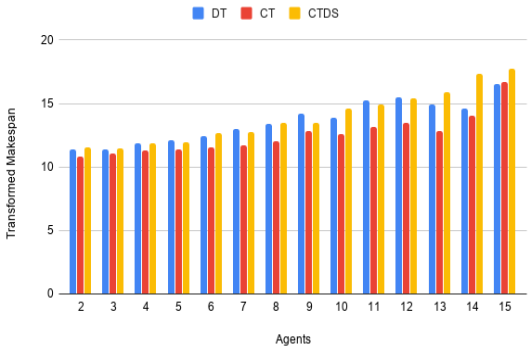

Figure 6.39: DFS Transformed Makespan

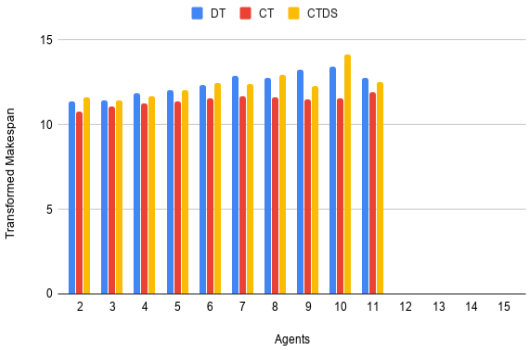

Figure 6.40: BFS Transformed Makespan

Target system makespan for all models when using BFS is presented in Figure 6.40 and DFS is presented in Figure 6.39. Note that the makespan is similar between themodels and the absence of values in the BFS graph, caused by the higher failure rate.

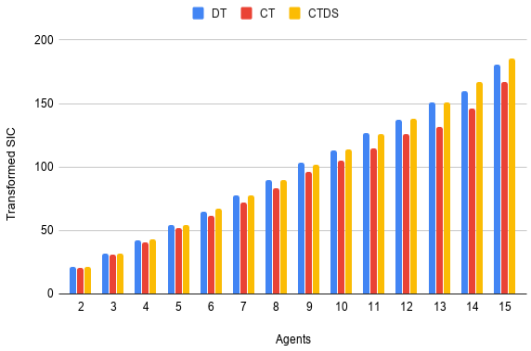

Figure 6.41: DFS Transformed SIC

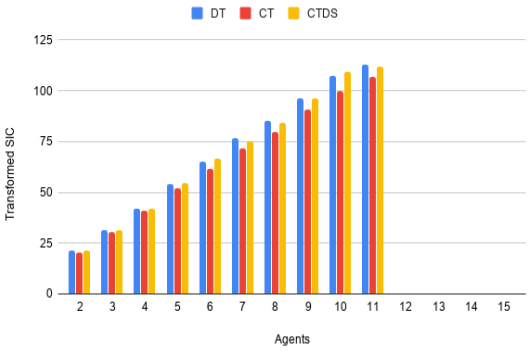

Figure 6.42: BFS Transformed SIC

Target system SIC for BFS and DFS is presented in Figures 6.41 and 6.42 respectively. Note that the y-axis differs between graphs. We observe an increase in SIC when using DFS instead of BFS, as expected. Note that problems with higher agent counts can be solved using DFS.

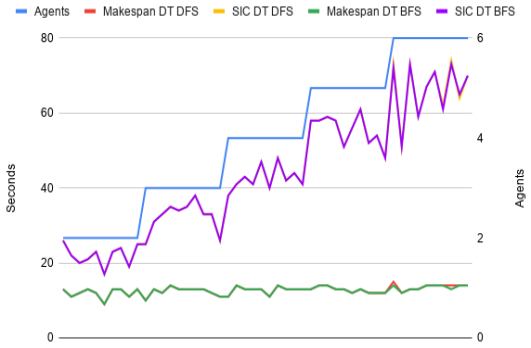

Figure 6.43: DFS and BFS costs, 6 agents.

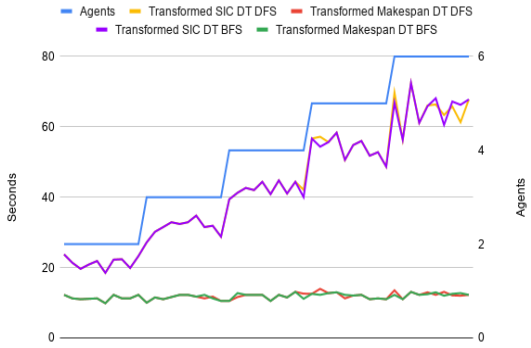

Figure 6.44: DFS and BFS target system costs, 6 agents.

For a final visualization of the BFS and DFS comparison, Figures 6.43 and 6.44 presents the costs both in the original models and the target system. We observe that the differences between the two methods are negliable.

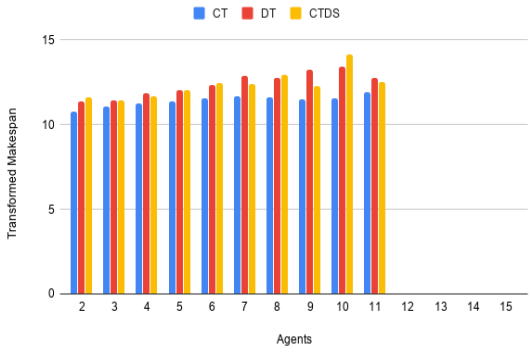

Figure 6.45: Admissible Transformed Makespan

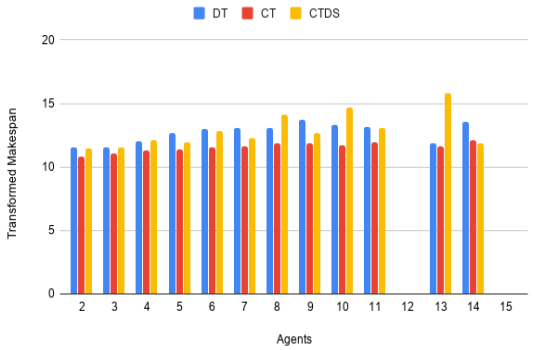

Figure 6.46: Pessimistic Transformed Makespan

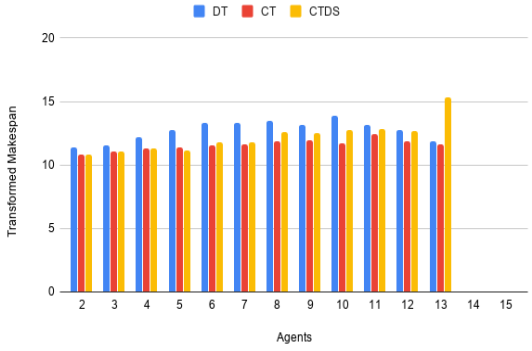

Figure 6.47: Very Pessimistic Transformed Makespan

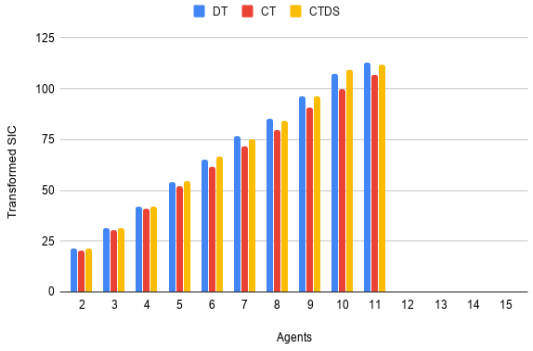

Figure 6.48: Admissible Transformed SIC

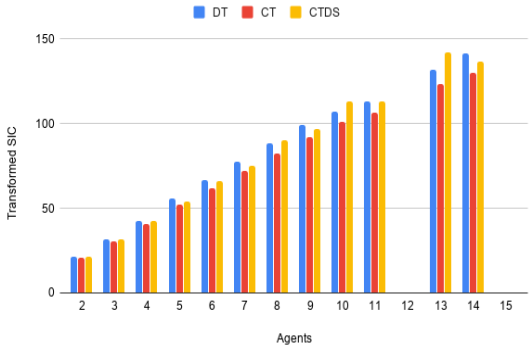

Figure 6.49: Pessimistic Transformed SIC

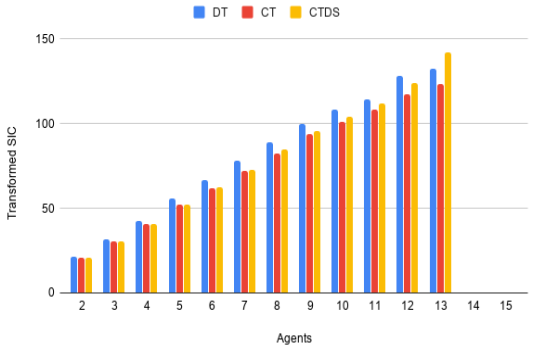

Figure 6.50: Very Pessimistic Transformed SIC

Figures 6.45,6.46 and 6.47 present the makespan of experiments using the different heuristics and Figures 6.48,6.49 and 6.50 present the SIC of experiments using the different heuristics. We observe that the costs are similar for all search configurations, but the admissible search tends to have a lower success rate.

6.5 Efficiency of transformation from abstract model to target system

Here we present the time required to transform a solution from an abstract model to the target system. We expect the transformation time to increase as more agents are added. However, the transformation time should be consistent between models as the abstraction level is not a consideration in this method. As presented in Figure 6.51, the choice of model appears to not have an effect on the transformation time, but the number of agents does have an effect. We note that the time scale of this experiment is different from the other results presented in this chapter, as the highest observed transformation time here does not reach half a second.

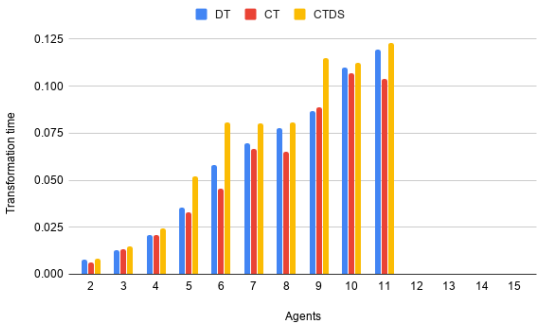

Figure 6.51: Abstract model to target system, transformation time.

6.6 Efficiency of abstract models at target system

In this section we present the results of evaluating all models on all graphs. First we present the runtime and then the costs at target system. Finally we present a summary of the results, to answer in which scenarios each model performs the best.

6.6.1 Runtime

Runtimes for the highway, highway entry and highway exit graphs are presented in Figures 6.52, 6.53 and 6.54. We observe that the CT model has the lowest runtime. Looking at Figure 6.52 we observe that the CTDS model has the second lowest runtime but in Figure 6.53 and 6.54 the DT model seems to have a lower runtime than the CTDS model.

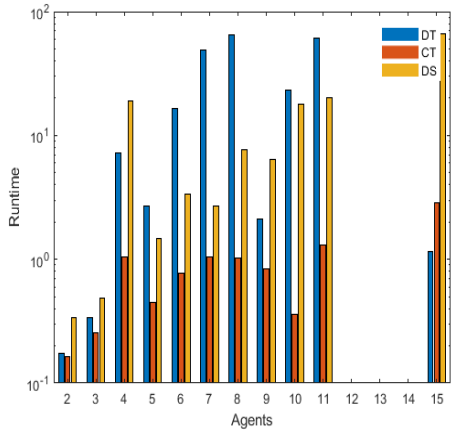

Figure 6.52: Highway runtime.

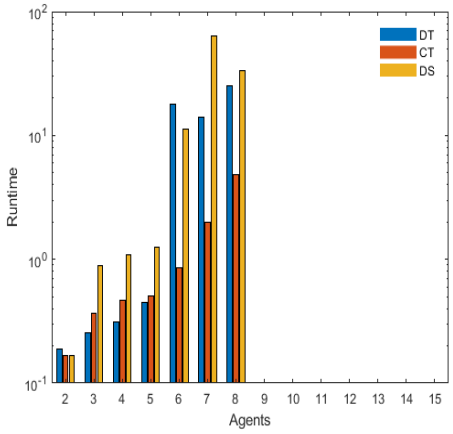

Figure 6.53: Highway entry runtime.

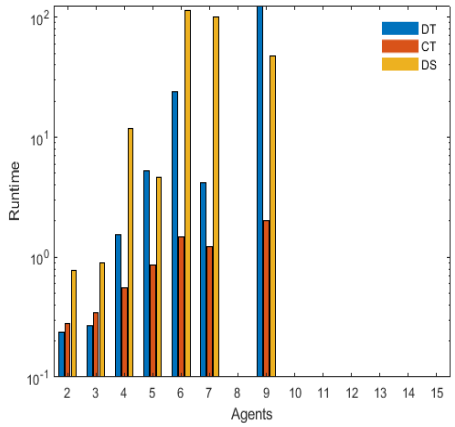

Figure 6.54: Highway exit runtime.

Runtime for the intersection graph is presented in Figure 6.55, we observe that the DS model is the first to produce higher runtimes at low agent counts, followed by the DT model. The CT model is able to solve problems with the highest agent counts. Due to the high variation of results, it is hard to decide which model produces the highest runtime.

The roundabout runtime is presented in Figure 6.56. We observe that the CT model is able to solve every problem with a runtime that is close to zero, while the DS model struggles to solve problems with around 9 agents. We observe that the DT model also has a runtime close to zero up to 8 agents but grows more beyond that.

The most curious result of the roundabout runtime experiment is the continuous time model, as it avoids all collisions. The exact reason for this behaviour is unknown, our hypothesis is that the specific structure and proportions of the roundabout graph interferes with the behaviour of the CT model.

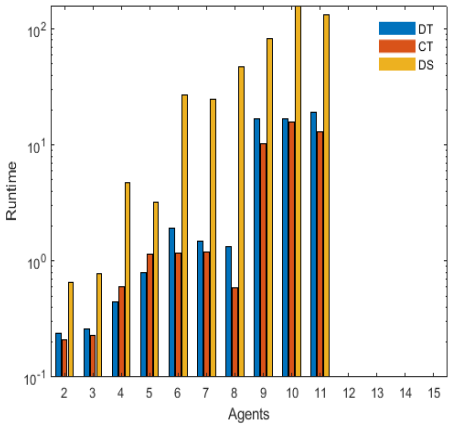

Figure 6.55: Intersection runtime.

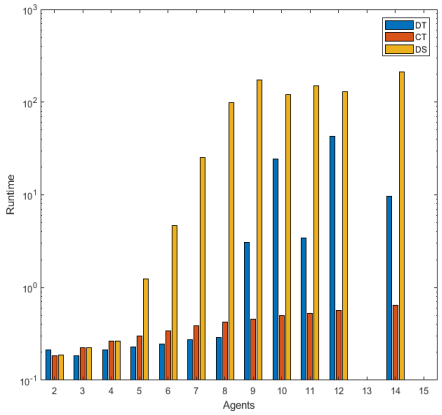

Figure 6.56: Roundabout runtime.

The runtime on the sparse graph is presented in Figure 6.57. We observe that the runtime initially grows in a linear fashion. This behaviour is explained by there being no collisions occurring at all, the observed runtime is that of solving a single node in the constraint tree. We observe that the DT model has the lowest runtime, this is due to the reduced complexity of the low level compared to the other models.

The runtime of the dense graph is presented in Figure 6.58. For agent counts of under 10, the DT model often has the lowest runtime. However, when there are more than 10 agents the CT model has the lowest runtime in the majority of cases. The explanation for this result is most likely that the CT model is more efficient than the DT model at resolving conflicts.

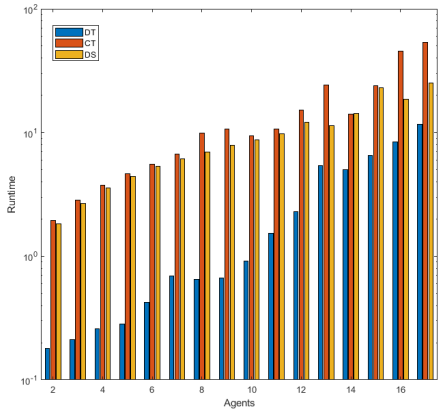

Figure 6.57: Sparse map runtime.

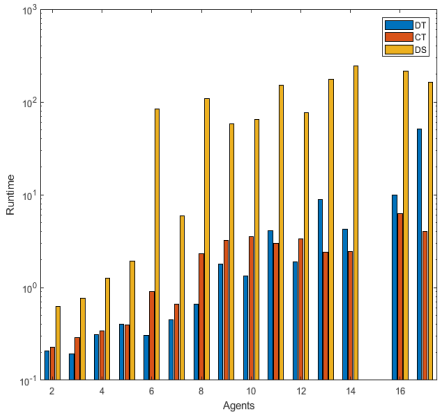

Figure 6.58: Dense map runtime.

6.6.2 Costs

In Figure 6.59 we present the SIC for each model in the intersection experiment. We observe that the SIC is similar between the models but CT is consistently the lowest. Figure 6.60 presents the makespan for each model in the intersection model, we observe that the makespan is generally lower for the CT model followed by the CTDS model.

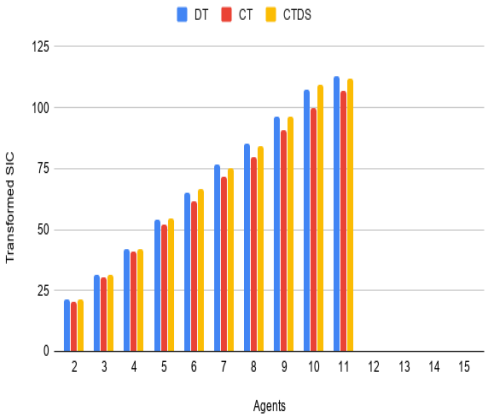

Figure 6.59: Intersection target system SIC.

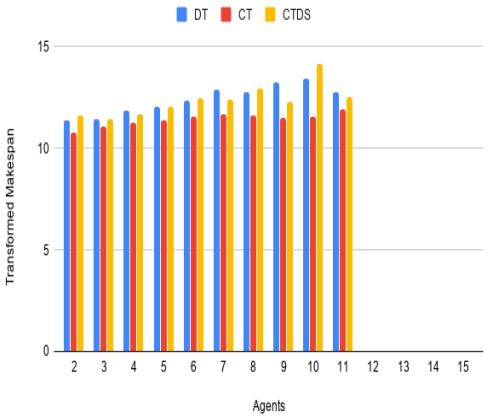

Figure 6.60: Intersection target system makespan.

Figure 6.61 presents the SIC in the roundabout graph. Observe that the results are similar, but the CT model often has a lower cost.

Figure 6.62 present the makespan for the roundabout graph. Again, the CT model has a lower cost

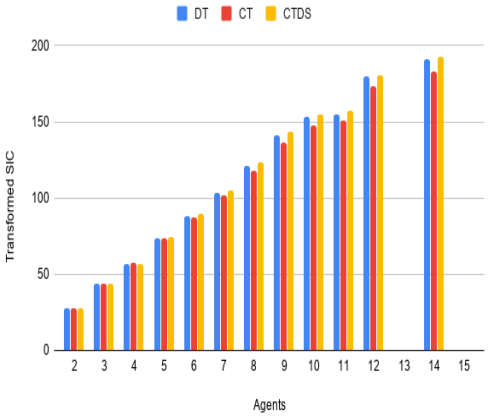

Figure 6.61: Roundabout target system SIC.

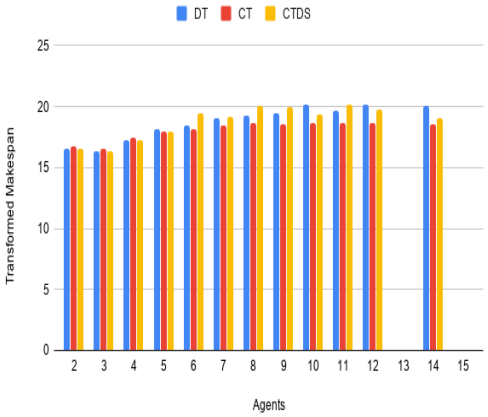

Figure 6.62: Roundabout target system makespan.

Figures 6.63 and 6.64 present the SIC and makespan for the highway graph. The CT model consistently has an equal or lower cost than competing models.

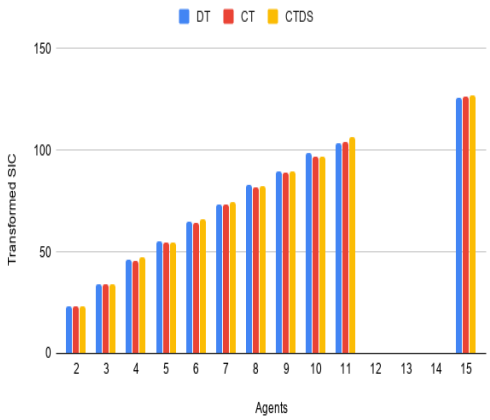

Figure 6.63: Highway target system SIC.

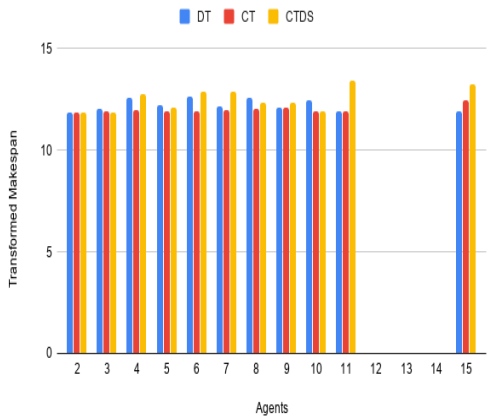

Figure 6.64: Highway target sytsem makespan.

In Figures 6.65 and 6.66, costs for the highway entry graph are presented. The CT model consistently has a lower cost.

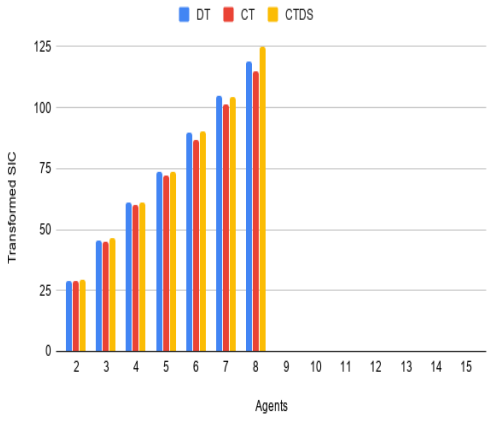

Figure 6.65: Highway entry target system SIC.

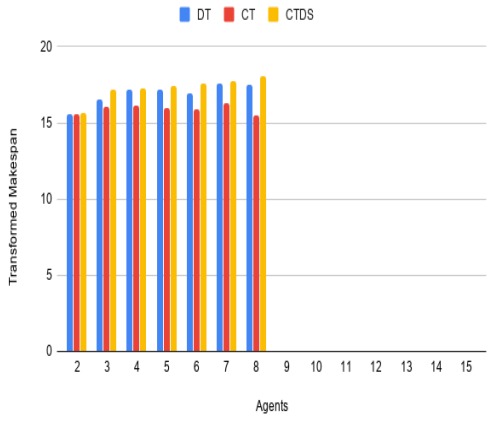

Figure 6.66: Highway entry target system makespan.

In Figures 6.67 and 6.68, costs for the highway exit graph are presented. The CT model consistently has a lower cost.

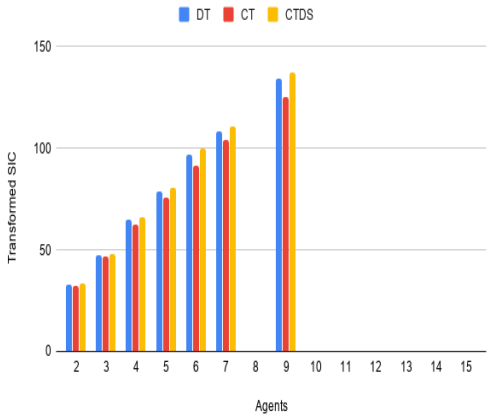

Figure 6.67: Highway exit target system SIC.

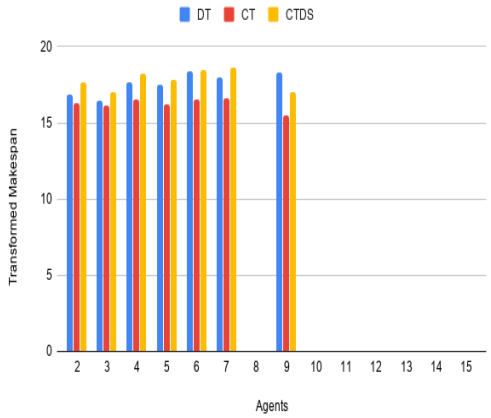

Figure 6.68: Highway exit target system makespan.

Figures 6.69 and 6.70 present the costs of the sparse graph. Observe that the DT model consistently has the highest cost, while the DS model has the lowest.

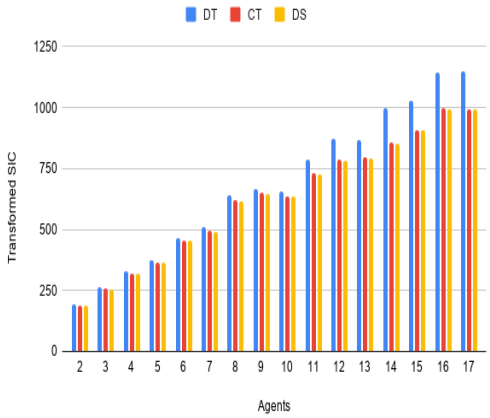

Figure 6.69: Sparse map target system SIC.

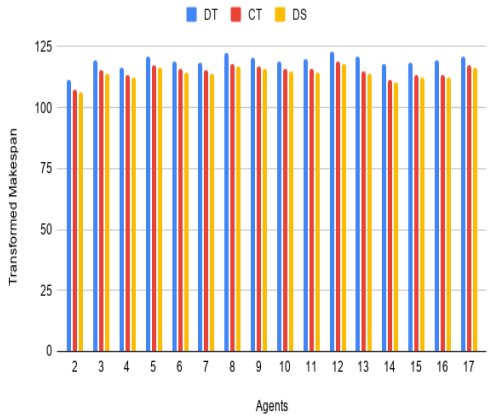

Figure 6.70: Sparse map target system makespan.

Figures 6.71 and 6.72 present the costs of the dense graph. Again, we see that the DT model has the highest cost while the CT and DS models are close to equal.

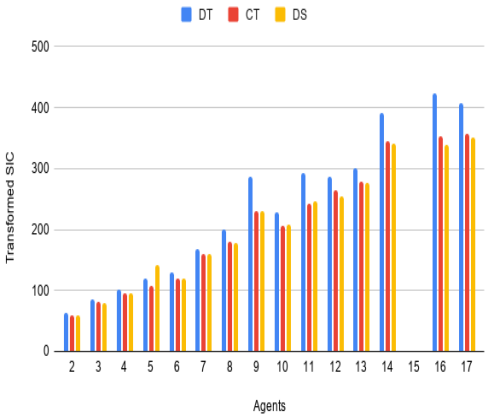

Figure 6.71: Dense map target system SIC.

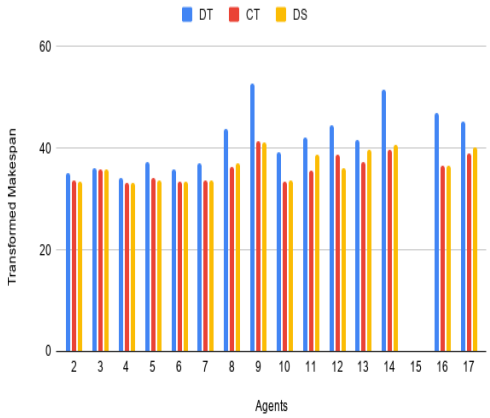

Figure 6.72: Dense map target system makespan.

In Table 6.2 we present what models have the lowest runtime and costs for each graph as a summary for the results of this section. We observe that the CT model is the best choice for most applications. We expected the CTDS model to produce better solutions at the cost of a longer runtime. Our hypothesis for this behaviour is that the conversion to target system does not consider arrival speeds at vertices, only arrival times, and that this does not always produce the best solutions. However, the CTDS model does consider the arrival speed. In other words, the CTDS model considers more factors than the enhanced MAPF-POST heuristic does.

Experiment	Runtime	SIC	Makespan
Intersection	CT	CT	CT
Roundabout	CT	CT	CT
Highway	CT	CT	CT
Highway entry	CT	CT	CT
Highway exit	CT	CT	CT
Sparse	DT	CTDS	CTDS
Dense	CT	CT & (CTDS <5 agents)	CT & (CTDS <5 agents)

Table 6.2: Summary results, all models run on all graphs.

7
Discussion

Our first research question asks how to convert a problem from the target system to its abstract representation in the considered models. From the results we can see how the required processing time grows, in a seemingly linear fashion, with the input data size. We see this eventually posing a problem when large maps are considered. However, as this data can be processed independently, before the problem instance is considered, pre processing the entire considered area is an attractive solution. However, the time consuming labour of manually cleaning up the graph is a large problem and not a viable option. The graph needs to be processed to improve the quality of it or a better source of data must be found.

Our second research question asks how to implement the algorithms to be efficient, considering both sequential and multi-core implementations. From the results we see that the multi-core implementation does not consistently reduce the runtime. The runtime difference appears to depend on several factors, such as what model is being used and how many agents are being used. The discrete-time model sees an increase in runtime when the multi-core solver is being used, while the models utilizing continuous time sees a reduction in runtime for higher agent counts. This is likely due to different computational demands of the low-level, and overhead introduced by synchronization of shared resources in the parallel version.

The third research question considers how the introduction of suboptimal methods affect the runtime of the abstract models and the solution costs at the target system. The results show how the runtime is reduced for all considered models and all considered numbers of agents when the DFS high level search is used. This in turn increases the success rate, more experiments would have to be performed using more agents to evaluate when the DFS search is unable to find solutions. We also see how using DFS search has a negligible effect on costs, both in the abstract models and the target system. Using suboptimal low-level heuristics did reduce the runtime in the CTDS model but had mixed results in the DT and CT models. The resulting costs in the abstract models were unaffected for the CT model, however the SIC increased for the DT and CTDS models. The target systems cost was unaffected for the CT model, but increased for the DT and CTDS models.

The fourth research question asks how solutions produced with abstract models perform at the target system, and which of them are the best choice in different scenarios. We see that while the solution costs varies very little between models, they are often very similar. When there is a difference, the CT model often produces the lowest . The runtime and success rate varies more between the different models, with the CT model often achieving the lowest runtime. Given these results, the CT model is the best choice in most scenarios. One notable exception to this is

the sparse map experiment where the DT model had the lowest runtime and the CTDS model had the lowest cost. However, as the focus of this work is the test case experiments, we consider those results to be the most significant of this work.

Related to the fourth research question is the efficiency of transforming a solution from an abstract model to the target system. The results show that the overhead added from the transformation is negligible when compared to the time required to solve problems. The runtime was tracked for every experiment, and the presented results are representative for all of them. We observed no runtime in excess of one second, while the solvers timed out after 5 minutes of runtime frequently.

7.1 Future work

Future work related to this project would be improving the heuristic to remove the presented limitations and maybe even claim optimality, using the provided paths from the abstract models. The multi-core implementation could be improved by reducing the overhead introduced by the shared resource, possibly by using the producer-consumer model. Taking parallelism to the next step could be by using a GPU to run the low-level search. Further speedup of the solver could be achieved by caching paths in the constraint tree. As it is now, all paths are recalculated each time a high level node is opened, even though only one path has changed from the parent node. However, keeping a path cache with limited size could lower the runtime by eliminating the need to calculate all paths every time.

We note the existence of algorithms for cooperatively facilitating safe manoeuvres [4, 15, 17, 7, 14, 8, 6]. Thus, further future work can include the implementation of the proposed solutions using simulations approaches, see [5, 11, 2].

Bibliography

[1] Anton Andreychuk, Konstantin Yakovlev, Dor Atzmon, and Roni Stern. Multi-agent pathfinding with continuous time. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 39–45, 2019.

[2] Christian Berger, Erik Dahlgren, Johan Grunden, Daniel Gunnarsson, Nadia Holtryd, Anmar Khazal, Mohamed Mustafa, Marina Papatriantafilou, Elad Michael Schiller, Christoph Steup, Viktor Swantesson, and Philippas Tsigas. Bridging physical and digital traffic system simulations with the gulliver test-bed. In Marion Berbineau, Magnus Jonsson, Jean-Marie Bonnin, Soumaya Cherkaoui, Marina Aguado, Cristina Rico Garcia, Hassan Ghannoum, Rashid Mehmood, and Alexey V. Vinel, editors, *Communication Technologies for Vehicles, 5th International Workshop, Nets4Cars/Nets4Trains 2013, Villeneuve d'Ascq, France, May 14-15, 2013. Proceedings*, volume 7865 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2013.

[3] Antonio Casimiro, Emelie Ekenstedt, and Elad Michael Schiller. Membership-based manoeuvre negotiation in autonomous and safety-critical vehicular systems. *arXiv preprint arXiv:1906.04703*, 2019.

[4] António Casimiro, Emelie Ekenstedt, and Elad Michael Schiller. Self-stabilizing manoeuvre negotiation: The case of virtual traffic lights. In *38th Symposium on Reliable Distributed Systems, SRDS 2019, Lyon, France, October 1-4, 2019*, pages 354–356. IEEE, 2019.

[5] António Casimiro, Jörg Kaiser, Johan Karlsson, Elad Michael Schiller, Philippas Tsigas, Pedro Costa, José Parizi, Rolf Johansson, and Renato Librino. Brief announcement: KARYON: towards safety kernels for cooperative vehicular systems. In Andréa W. Richa and Christian Scheideler, editors, *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*, volume 7596 of *Lecture Notes in Computer Science*, pages 232–235. Springer, 2012.

[6] António Casimiro, Jörg Kaiser, Elad Schiller, Pedro Costa, José Parizi, Rolf Johansson, and Renato Librino. The KARYON project: Predictable and safe coordination in cooperative vehicular systems. In *43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop, DSN Workshops 2013, Budapest, Hungary, June 24-27, 2013*, pages 1–12. IEEE Computer Society, 2013.

[7] António Casimiro, Oscar Morales Ponce, Thomas Petig, and Elad Michael

Schiller. Vehicular coordination via a safety kernel in the gulliver test-bed. In *34th International Conference on Distributed Computing Systems Workshops (ICDCS 2014 Workshops), Madrid, Spain, June 30 - July 3, 2014*, pages 167–176. IEEE Computer Society, 2014.

[8] António Casimiro, José Rufino, Ricardo C. Pinto, Eric Vial, Elad Michael Schiller, Oscar Morales Ponce, and Thomas Petig. A kernel-based architecture for safe cooperative vehicular functions. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014, Pisa, Italy, June 18-20, 2014*, pages 228–237. IEEE, 2014.

[9] Johan Gerdin. Multi-agent pathfinding with discrete speeds. Master's thesis, Computer Science and Engineering, Chalmers University of Technology, Rännvägen 6B, Göteborg, Sweden, S-412 96, 6 2020. To appear.

[10] Wolfgang Hönig, TK Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

[11] Mitra Pahlavan, Marina Papatriantafilou, and Elad Michael Schiller. Gulliver: A test-bed for developing, demonstrating and prototyping vehicular systems. In *Proceedings of the 75th IEEE Vehicular Technology Conference, VTC Spring 2012, Yokohama, Japan, May 6-9, 2012*, pages 1–2. IEEE, 2012.

[12] Thomas Petig, Elad M Schiller, and Jukka Suomela. Changing lanes on a highway. In *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[13] Mike Phillips and Maxim Likhachev. SIPP: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, May 2011.

[14] Oscar Morales Ponce, Elad Michael Schiller, and Paolo Falcone. Cooperation with disagreement correction in the presence of communication failures. In *17th International IEEE Conference on Intelligent Transportation Systems, ITSC 2014, Qingdao, China, October 8-11, 2014*, pages 1105–1110. IEEE, 2014.

[15] Oscar Morales Ponce, Elad Michael Schiller, and Paolo Falcone. How to stop disagreeing and start cooperatingin the presence of asymmetric packet loss. *Sensors*, 18(4):1287, 2018.

[16] Aleksander Sadikov and Ivan Bratko. Pessimistic heuristics beat optimistic ones in real-time search. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 148–152. IOS Press, 2006.

[17] Vladimir Savic, Elad Michael Schiller, and Marina Papatriantafilou. Distributed algorithm for collision avoidance at road intersections in the presence of communication failures. In *IEEE Intelligent Vehicles Symposium, IV 2017, Los Angeles, CA, USA, June 11-14, 2017*, pages 1005–1012. IEEE, 2017.

[18] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent path finding. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012.

[19] Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

A

Appendix 1