# Team 15

# Police Thief Chase

by

Marvin Boltes

boltes@kth.se

Lukas Konwitschny

lukasko@kth.se

Raghunath Vairamuthu

rvai@kth.se

Mukund Seethamraju

mukund@kth.se

## Abstract

In everyday life there are situation where one person has to catch someone or something while both instances are moving. One example would be the police trying to catch a thief who is on the run with some stolen goods. For common informed search algorithms like the A* algorithm this is a hard task and it costs a lot of time to compute a new route every time the situation changes. In this report we introduce and make use of the Moving Target D* Lite algorithm, that is designed for exactly this problem. After the instruction of the MTD* Lite follows our case study where we make use of the algorithm and discuss the results. Next we will compare the MTD* Lite to the common A* and point out the potential of the algorithm.

# CHAPTER 1

## Introduction

The Routing Problem in a given map is a well researched issue. For example every navigation software for a car or phone solves this task with ease. But in most cases the destination is fixed and doesn't change over time. That's the reason why it can be computed so fast, because even if the path has to change slightly (e.g. traffic or blocked roads), a lot of the path is still known and this information can be used. In our case the target can change over time, what will make it a heavier problem to efficiently compute a perfect path to the destination.

In our case study we want to simulate a car chase where a police car has to catch a thief's car. For this scenario we want to use a simplified map of Stockholm and fictional traffic information as underlying environment. The thief starts at a special starting point and tries to drive to a certain goal position. A police car starts at another starting point and tries to catch the thief's car. As a result, the thief car has a fix destination point, whereas the goal of the police (the moving thief car) is changing. Because of time and efficiency reasons the police has to use a quick routing planner to catch the thief, otherwise he will be able to escape Stockholm. Classic path planning algorithms will fail to calculate the perfect way in an acceptable time. The map of Stockholm will be modeled as a graph, with different parts and highlights of the city as nodes and the connection between them as edges. The distance between the nodes will be represented by the actual distance in the city of Stockholm (see appendix). Moreover, we take into account that the police is faster than the thief. Therefore, we multiply the cost (the distance) of each edge with a normally distributed value that models the traffic. The normal distribution for the police has a lower mean than the one for the thief to achieve a higher pace for the police. In our case study we assume that the police always knows the position of the thief (e.g with the help of traffic cameras or GPS tracker).

Besides our case study the Police-Thief-Chase, there are many different fields of application. For Example a medicine in the body trying to catch an infected cell.

# CHAPTER 2

# Related work

A moving target search problem can be addressed in two different ways. One is an offline approach [MS09] which takes into account all possible stakes in picture, like changes in environment, target move list etc. It may be computationally more expensive than an online approach. Online approaches like [Sch03], on the other hand, combine planning and movements effectively to optimize for smaller computation times. Real-time search algorithms employ limited look-ahead of each search and hence are not affected much by the total number of possible states. There is also an improvement to the above mentioned algorithms in the form of incremental search where the previous search tree can be transformed to the current search tree and the search starts by using the old data instead of staring from scratch. Generalized adaptive A* [XY08], Differential A* and D* Lite follow this approach. In [SK10], Moving Target D* lite is used, which is a form of incremental search algorithm that is up to seven times more efficient than the traditional A* algorithm. It uses the principle behind Generalized Fringe-Retrieving A* (G-FRA*). G-FRA* is an incremental search algorithm that generalizes Fringe Retrieving A* (FRA*) from grid worlds to arbitrary directed graphs. On the other hand, D* Lite solves sequences of search problems in dynamic environments where the start state does not change over time by repeatedly transforming the previous search tree to the current search tree. Interleaving these two algorithms delivers very fast search results, according to [SK10].

Based on the findings in [SK10] we decided to use MTD* Lite to solve our moving target search problem, as it looks like this is the most efficient of the above mentioned algorithms.

# CHAPTER 3

# Methods and Implementation

In order to solve the moving target search problem that occurs in our use case, we followed two approaches. On the one hand we used a simple A* algorithm to compute a new path every time the target moves, on the other hand we used a specialized algorithm for moving target search, namely the MTD* Lite algorithm. By doing this, we can analyze how the specific algorithm compares to a naive solution.

## 3.1 A* Solution

In this naive solution approach we use the A* algorithm repeatedly to compute the new optimal path between the current position of the police and the new position of the thief. Every time there should be computed a new path, the A* is initialized and starts the search from scratch. This solution should form a basis for the analysis as it is a kind of brute force solution for the moving target search problem.

## 3.2 MTD* Lite Solution

This more sophisticated approach uses an extended A* algorithm which is called MTD* Lite which specializes A* for moving target search. In this solution we therefore try to solve the problem in a more efficient way than the naive solution. The main difference between the MTD* Lite and A* is that MTD* Lite is initialized only once at the beginning of our simulated catch scenario. Every time a new path should be computed, because the police and/or thief has moved, the algorithm changes the start and goal node and computes the new path. It does not initialize the whole algorithm every time and uses knowledge of search trees of previous searches to calculate the new path. That is the reason why this advanced version of the A* should be more efficient. A rough overview how the MTD* Lite

algorithm works is given in the next section, for a more detailed description please refer to [SK10].

## 3.3 MTD* Lite Algorithm

MTD* Lite is an incremental search algorithm that extends A* in order to solve moving target search problems. As A* it stores a g-value, f-value, h-value and a parent pointer for each node. As in A* the g-value is an approximation of the costs from the search start to this node, the h-value is an approximation of the costs from this node to the goal node and f-value is the sum of g and the f-value and thus an approximation of the costs of a path from start node to end node by using this node. In our project we used the distance between two nodes as h-value as well as for the weight of an edge. In addition to these four values the MTD* Lite stores a rhs-value that is a look ahead g-value and is defined as follows:

$$rhs(s) = \left\{ \begin{array}{ll} c & \text{if } s = s_{\text{start}} \\ \min\limits_{s' \in Pred(s)}(g(s') + c(s', s)) & \text{otherwise} \end{array} \right\}$$

The first run of the MTD* Lite is quite similar to a run of A*, as it expands nodes according to their h-values until it finds the goal node. During the expansions, all the values of a node get updated. After the target has moved in the graph, the algorithm is called again to compute a path to the new target node. Therefore, all already expanded nodes that are not in the sub tree rooted by the new search start node get closed, i.e. their values are reset, as these nodes could be reached with lower cost from another node in the new setting . In contrast to the A* algorithm, all other expanded nodes are not modified and thus their values are used to compute the new path. A detailed description and pseudo code is delivered in [SK10]

## 3.4 Implementation of the Use Case

The first step for realizing our use case was to create a scenario with a graph of places in Stockholm, a start point for the police and a start point for the thief. This information was encoded in a simple text file which is the input for our simulation.

In the simulation we execute the following steps:

1. Parse the text input file to retrieve the information needed for the simulation
2. Decide which of the two solutions should be used according to a program argument
3. Calculate the fix path for the thief with a normal A* algorithm

4. Initialize the algorithm for the moving target search
5. Simulate the catch in a loop until the thief is caught or the thief reached his goal node
   - Every time the thief reaches a node, select the next node out of the thief path and let the thief go there. The time needed for this part is computed as *distance * traffic for thief*
   - Every time the police reaches a node, check if the thief is caught (i.e. the police is where the thief is heading or where the thief is right now). If it isn't caught and the thief position changed, compute a new path for the police with the selected algorithm and take the first step of this new path. The time needed for this part is computed as *distance * traffic for police*
   - Decrement the time variables for police and thief (simulate elapsed time)

A more detailed description of our simulation is given in algorithm in the appendix. Our simulation produces a text output file that contains information about the simulated catch. This file can be used to create a visualization of the simulation with our Matlab program.

# CHAPTER 4

# Analysis

As mentioned in chapter 3 the naive approach of a repeated application of the A* algorithm delivers optimal paths for the moving target search problem. This holds as the A* algorithm returns an optimal path for a search problem with a static target and in our case of a moving target the problem is divided into several sub problems with a static target.

Thus, the selected specified moving target algorithm does not improve the result of the moving target search problem, but does not worsen it either, since this algorithm delivers the optimal paths as well. The aim of the MTD* is rather to solve the moving target problem in a more efficient way than the A* algorithm. As explained before, it therefore uses already expanded nodes in the search tree instead of building the whole tree from scratch for every changing target position.

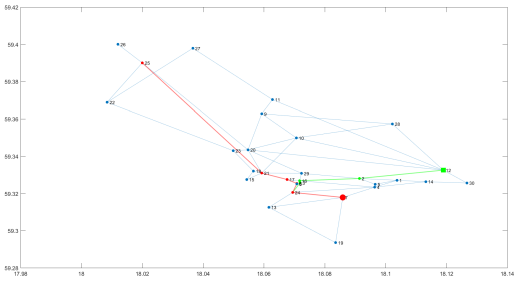


***Fig. 4.1:*** *First step of the catch simulation*

In figures 4.1, 4.2 and 4.3 the first three steps of a sample simulation are visualized. The the red point refers to the thief position whereas the green point marks the police. The planned paths are also colored in the corresponding color. Here one can see how the police adjusts its path to the movement of the thief.

In order to compare the amount of nodes that are expanded during the execution of both variants of moving target search solvers, we counted this amount during the same

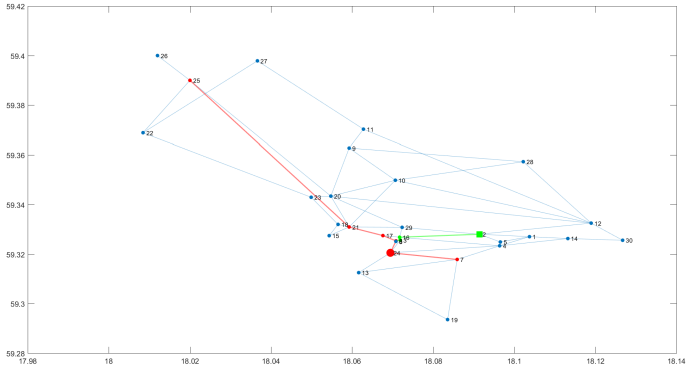



***Fig. 4.2:*** *Second step of the catch simulation*

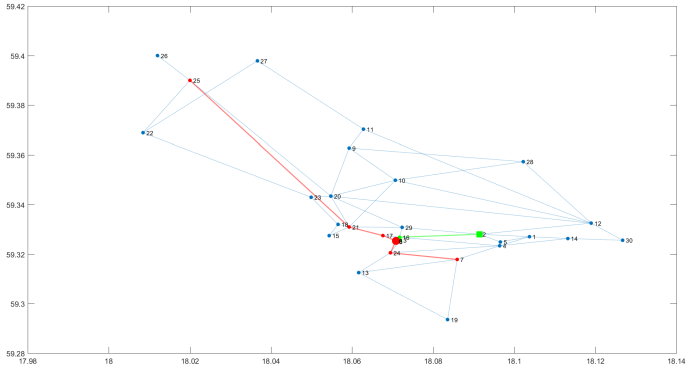



***Fig. 4.3:*** *Third step of the catch simulation*

scenario. Even in our relatively small test case one can recognize a significantly smaller amount of node expansions when using MTD* in contrast to when using multiple A*. As shown in figure 4.4, in our case the MTD* expanded 7 nodes while the A* expanded 11. Hence, the MTD* expanded 36.4% less nodes than A*. These values were obtained with a standard derivation of 0 to guarantee comparable results for the two runs. Of course the exact values depend on the starting node for the police and the path the thief follows. In larger problem instances with more nodes and longer paths to the target this effect is expected to be even stronger. In these cases the MTD* saves more expansions as the A* algorithm has to expand a lot more nodes during it's search whereas the MTD* is likely to find a path in the already expanded sub search tree.

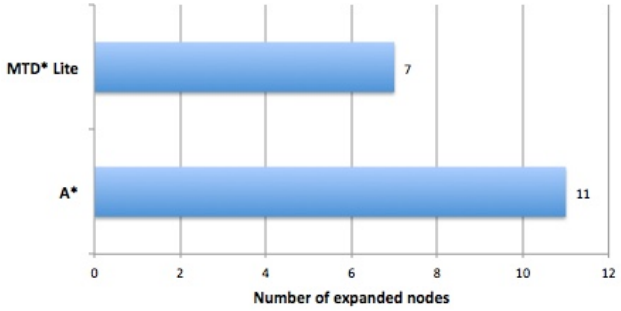



***Fig. 4.4:*** *Expanded nodes of MTD* Lite and A**

This reduction in expansions on the one hand reduces the space complexity, as the nodes stored in the *open lists* are less, and on the other hand leads to a faster run time, since the expansion of nodes is the most time consuming operation of the search algorithms. In our small example the difference in the run time is not observable which is probably cased by other time consuming operations in the execution of the program and the small difference of run time for the two algorithms in the small problem instance. Similar to the node expansion, also the difference in the run time is expected to grow with the size of the problem instance. Therefore, in large instances it should be clearly observable that the MTD* runs faster than the A*. This is also proved by the results of the analysis in [SK10] (figure 4.5).

| | searches per test case | moves per test case | expanded states per search | deleted states per search | runtime per search |
|---|---|---|---|---|---|
| A* | 691 | 691 | 14489 (24.1) | | 6043 |
| Di↑erential A* | 691 | 691 | 14489 (24.1) | | 8385 |
| GAA* | 691 | 691 | 11246 (20.3) | | 4432 |
| Basic MT-D* Lite | 690 | 690 | 913 (5.2) | | 917 |
| MT-D* Lite | 690 | 690 | 535 (4.5) | 550 (18.4) | 570 |

***Fig. 4.5:*** *Results from the analysis in [SK10]*

Regarding the implementation of the two different algorithms one can say that the MTD* requires more implementation effort than the A*, as it is an extension of it. Nevertheless, the implementation effort stays reasonable considering the gain of efficiency.

Like assumed from the beginning, the result of a catch, i.e. if the thief has been caught or not and if yes where, strongly depends on the starting point of the police, the path of the thief and the pace for police resp. thief, which is controlled by the parameters of the normal distributions for the traffic volume. If the police has a significantly higher pace than the thief it is more likely that it catches the thief.

Of course our test scenario can be improved by changes in the way it is modeled. In order to keep our use case simple for implementation and analysis we made several assumptions that make the model more unrealistic. We tried to model the traffic between two nodes with normal distributed traffic factors. Thereby we assumed that there is the same distribution of traffic in every edge of our graph which is of course not the case in reality. This could be improved by assigning different distribution parameters to each edge according to the real traffic volume. In addition, to gain an even more realistic model one can change the traffic volume for different times of a day. There is for example definitely more traffic in whole Stockholm between 16:00 and 19:00 than between 12:00 and 14:00 on working days.

Another simplifying assumption was that the thief follows a static path during the race and does not take traffic information or knowledge about the police position into account. The thief could for example try to avoid the police or to shake it off. Taking these two points into account would lead to a much more realistic, but also extremely more complicated problem.

As the weight of an edge we took the air-line distance between the two nodes of the edge, which is obviously not the real distance for a car to drive from one node to another, e.g. two crossroads, as there could be curves in between. A more sophisticated weight would result in a more realistic use case, but is hard to calculate. As the heuristic for the path cost from a node to the target was also generated from the air-line distance, this can be improved in a similar way to get a better performance of the algorithms.

In our use case the police permanently knows to which node the thief is heading. This is possible if the police installed a GPS tracker on the thief's car. If that's not possible the police has to deal with the information it gets from the patrol car*s* or from security cameras, which changes the problem and makes it harder to solve.

An interesting extension of our test case can be achieved by using multiple police cars. This would result on a multi agent problem on the police side and requires communication and coordination between the different cars. An approach for this problem is described in [Sch03].

# CHAPTER 5

## Summary

In our project we tried to find solutions for a moving target search problem. As use case for this problem we used a scenario where a police car tries to catch a thief car. The whole scenario takes place in a simplified map of Stockholm. Although we tried to model the use case as realistic as possible, there are several possibilities to improve and extend the model. For the calculation of the path for the police we used A* as well as the advanced version MTD* Lite. They both lead to the same result, but MTD* Lite is more efficient according to literature and the analysis of our application. This confirmed our assumption that a specialized algorithm is more suitable for this kind of problem than a naive repeatedly execution of A*.

# Bibliography


[MS09] C. Moldenhauer and N. Sturtevant: *Optimal solutions for moving target search*. In: (2009).

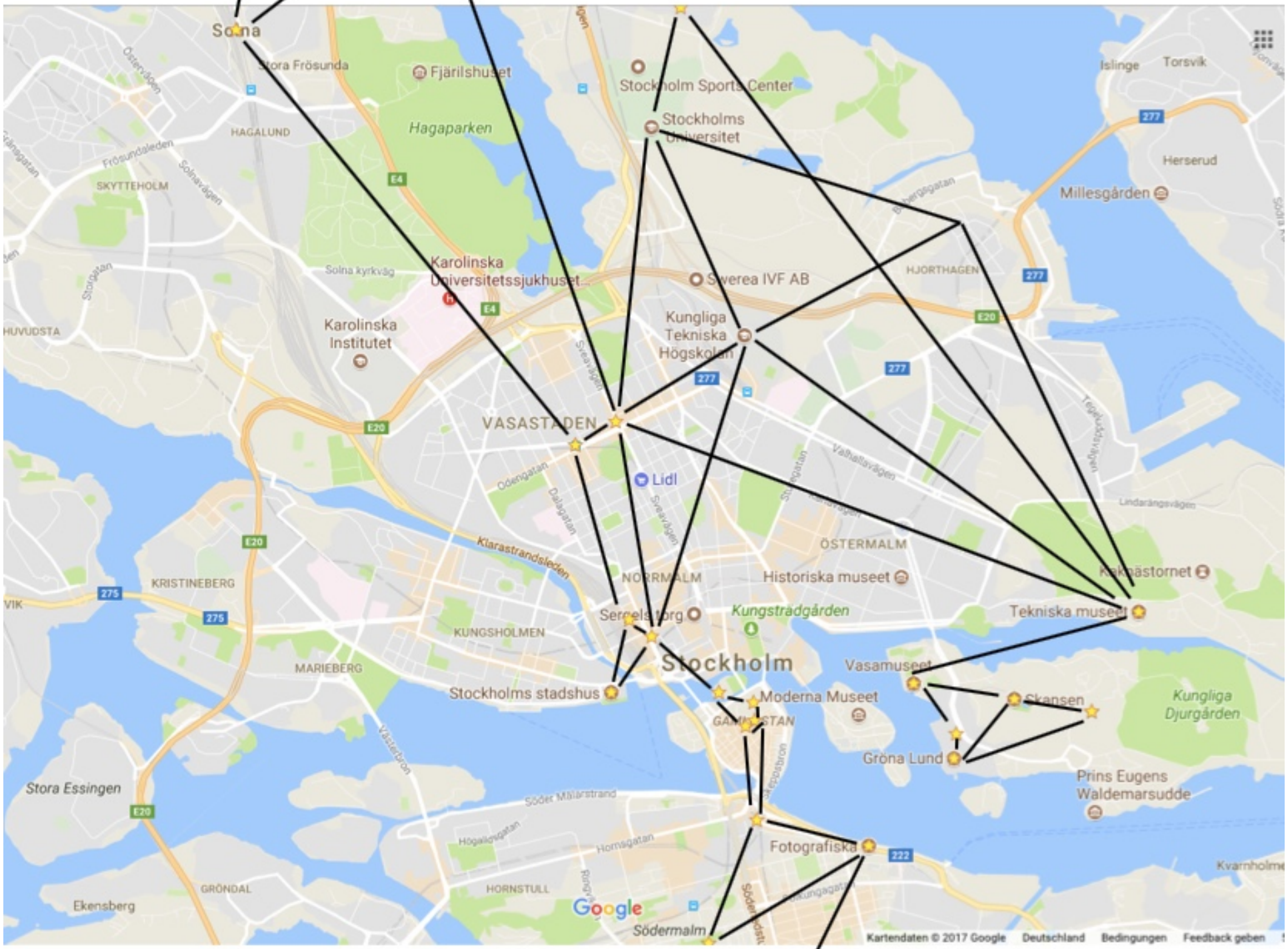
[Sch03] M. G. A. K. X. W. J. Schaeffer: *Multiple agents moving target search*. In: (2003).

[SK10] W. Y. X. Sun and S. Koenig: *Moving target D lite*. In: (2010).

[XY08] S. K. X. Sun and W. Yeoh: *Generalized Adaptive A**. In: (2008).

**Algorithm 1:** Algorithm for catch simulation

```
Define Graph
Define A* search algorithm:
    Input: Start node, End node, Graph with costs of edges
    Output: List of nodes which tells the path till the end node
Define MT D* algorithm
    Input: Start node or current node, Target node, Graph with costs
           of edges
    Output: List of nodes which tells the path till the Target node
Initiate the Graph
Define the player Police and starting node and the distribution of
 traffic associated with Police_car ND_P
Define the player Thief and his starting node and the distribution of
 traffic associated with Thief_car ND_T
Set the destination for the Thief
Set the timers Time, Time_P, Time_T to 0
Run the A* search algorithm for the Thief:
    Input: Start node, Destination node, Graph
    Output: Path till Destination in a list
Define variables for current nodes Current_P and Current_T
Define next node of the Thief, Next_T
Run the MT D* algorithm for the police:
    Input: Current_P,Next_T , Graph
    Output: Path from Current_P to Next_T
Update timers:
    Time_P = (Distance between current and next node of
 police)*(random number from ND_P)
    Time_T = (Distance between current node and next node of
 thief)*(random number from ND_T)
While (Current_P != Current_T & Current_T != Destination
 node):
    If (Time_P == 0):
        Update Current_P
        If (Current_P  != Next_T):
        Run MT D* algorithm for police:
                Input: Current_P,Next_T , Graph
                Output: Path from Current_P to Next_T
            Update Time_P
        Else if (Current_P  == Next_T):
        Break and report the WIN
    If (Time_T == 0):
        Update Next_T
        If (Current_T != Destination node):
            Run MT D* algorithm for police:
                Input: Current_P,Next_T , Graph
                Output: Path from Current_P to Next_T
            Update Time_T, Time_P
    Else if (Current_T == Destination node):
        Break and report Lose
```


Solna
Stora Frösunda
Fjärilshuset
Stockholm Sports Center
Stockholms Universitet
Islinge
Torsvik
Herserud
Millesgården
Hagaparken
HAGALUND
SKYTTEHOLM
Karolinska Universitetssjukhuset
Karolinska Institutet
Sverea IVF AB
HJORTHAGEN
Kungliga Tekniska Högskolan
VASASTADEN
Lidl
ÖSTERMALM
Historiska museet
Kaknästornet
Tekniska museet
Kungsträdgården
KRISTINEBERG
KUNGSHOLMEN
MARIEBERG
Stockholm
Stockholms stadshus
Moderna Museet
Vasamuseet
Skansen
Kungliga Djurgården
Gröna Lund
Prins Eugens Waldemarsudde
Stora Essingen
Fotografiska
GRÖNDAL
HORNSTULL
Ekensberg
Södermalm
Kvarnholmen
Google
Kartendaten © 2017 Google
Deutschland
Bedingungen
Feedback geben