UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

# Algorithms for Moving-Target TSP

*Algorithmen für bewegende Ziele in TSP*

**Bachelorarbeit**

verfasst am
**Institut für Theoretische Informatik**

im Rahmen des Studiengangs
**Informatik**
der Universität zu Lübeck

vorgelegt von
**Felix Greuling**

ausgegeben und betreut von
**Prof. Dr. Maciej Liskiewicz**

Lübeck, den 28. November 2019

IM FOCUS DAS LEBEN

## Eidesstattliche Erklärung

*Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.*

Felix Greuling

## Zusammenfassung

TODO

## Abstract

TODO

## Acknowledgements

TODO

# Contents

# 1 Introduction

## 1.1 Contributions of this Thesis

TODO

## 1.2 Related Work

TODO

## 1.3 Structure of this Thesis

TODO

# 2

# Moving-Target TSP in one dimension

This specific instance was introduced in [**helvig**]. Each target and the pursuer are restricted to a single line, so there are only two possible directions of movement. The first naive approach is to calculate the cost of the tour to intercept all the targets on the right side of the origin and then on the left and vice versa. Choose the tour with lower costs. This will probably result in an unbounded error with simple counter examples (TODO: create a graphic with a counter example), where the pursuer probably takes and eternity to intercept a target.

Therefore, the pursuer is only allowed to change his direction, whenever he intercepts the fastest target either on the left or right side of the origin. These targets are named as turning points. Turning points that are not the fastest target can not reduce the tour time. This time not used to catch up with the fastest target is equivalent to waiting at one point. As already mentioned (TODO), waiting at one point results in not optimal tours.

With this knowledge about turning points, the term *state* is introduced. A state is a snapshot of a tour. It represents a potential turning point at which the pursuer then attempts to reach either the next fastest target on the same or the other side of the origin. It required two targets to define a state. First, the target, where the pursuer is currently located ($s_k$), and second, the fastest target ($s_f$) on the other side of the origin. Thus, a state can be described as the a tuple $(s_k, s_f)$. As special cases, there are the states $A_0$ and $A_{final}$. Neither $A_0$ or $A_{final}$ have such a tuple $(s_k, s_f)$, these states just define the start and end of each tour. For each state $A_i$, the shortest time to reach the state can be calculated with the time function $t$. It applies $t(A_0) = 0$

In order to determine all states, it is necessary to divide the targets into the lists *Left* and *Right*. Each target, which is located on the left of the origin, is inserted in the Left list. Analogous for the list Right. Now the targets are sorted by the speed leading away from the origin in descending order. Targets that are closer to the origin and additionally slower than others are removed in the respective lists. Thereby, just the potential turning points are remaining. In order to determine the list of all states, all combinations of the lists Left and Right and vice versa are inserted for $s_k$ and $s_f$. The list is now sorted in ascending order of the sum of the indices of the targets from the lists Left and Right. Therefore, the combinations of the fastest targets are in front of the state list.

In a state A, there are two options: Either the fastest target on the left or on the right side of the pursuer (also from the perspective of the origin!) is intercepted next. These targets in turn are potential turning points, i.e. a *transition* into the next state B is generated. The notion of a transition between the states A and B is $A \rightarrow B$. With the options to move to the left or right turning point, there are two transitions outgoing the current state: $\tau_{left}$ and $\tau_{right}$. Thus, each state has up to two transitions into other states. As soon as each turning point is intercepted on the left side of the origin, there is only the transition $\tau_{right}$ into the final state, which is then named as $\tau_{final}$. The weight of the edge is then determined by time needed to intercept all remaining targets on the other side. The final state has no transitions into other states.

The time function $t$ can also be given transitions with $\tau_{left}$ or $\tau_{right}$. Thereby, the time is calculated to intercept the fastest target on the left or right side from the considered $s_k$ in state $A$ at time $t(A)$.

Now the problem with states and transitions between them can be transformed into a graph problem. The states represent the vertices $V$ and the transitions are the edges $E$ between the vertices. Thus, the Graph $G$ can be modelled as $G = (V, E)$. Next the exact structure of $G$ needs to be specified. First, the start state $A_0$ is inserted. As previously mentioned, the state list is generated in ascending order of the sum of the left and right indices. Now the states with the lowest sum value are placed next to $A_0$. Next, the same thing is done with the next higher sum value with its states placed next to the states of the underlying sum value. Proceed with this process for all other states until the end of the list and append $A_{final}$ at the end. Transitions can only lead to states of higher sum values. Therefore, the graph is acyclic, as there are no edges into the or previous vertices. Caused by some transitions immediately into $A_{final}$, there are probably vertices to which no edge is drawn.

The new graph problem can thus be solved as a shortest path problem by finding the shortest way from $A_0$ to $A_{final}$. Since the graph is acyclic, a simple procedure can be chosen. The authors of [**helvig**] have opted for dynamic programming. Their algorithm iterates through each state in topological order. By considering $\tau_{left}$ and $\tau_{right}$, check, whether the subsequent state can be reached faster than possibly before with a different sequence of states. Iterating the states squares the runtime, which requires the algorithm to run exactly $O(n^2)$.

# 3

# Conclusion

TODO