Aalto University
School of Science
Department of Mathematics and Systems Analysis

Joonas Haapala

Routing Military Aircraft by Solving a Dynamic Multi-Objective Network Optimization Problem with the A* Algorithm

Bachelor's Thesis

Espoo, June 8, 2015

Supervisor: Prof. Kai Virtanen

Instructor: M.Sc. Heikki Puustinen

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of bachelor's thesis

Author Joonas Haapala

Title of thesis Routing Military Aircraft by Solving a Dynamic Multi-Objective Network Optimization Problem with the A* Algorithm

Degree programme Degree programme in Engineering Physics and Mathematics

Major Systems Sciences **Code of major** F3010

Supervisor Prof. Kai Virtanen

Thesis advisor(s) M.Sc. Heikki Puustinen

Date 8.6.2015 **Number of pages** 25 **Language** English

Abstract

Flight paths of military aircraft can be optimized with respect to various criteria simultaneously. Methods exist for finding these paths by using Dijkstra's algorithm on a weighted network, where the weight of each edge is a weighted sum of criteria such as fuel consumption and threats caused by opponents. H. Puustinen describes one such solution in his Master's thesis. These methods, however, assume the weights remain constant for the duration of the flight and thus are unable to take into account the movement of time-dependent threats.

This thesis introduces a modified A* network search algorithm that is able to solve paths through a weighted network with time-dependent weights. It defines a new dominance relation for search nodes in the network, which is a vital component required to keep the algorithm running time within practical limits. The thesis also shows and compares some heuristic functions for the A* algorithm. The results from various scenario imply that the new algorithm is able to find paths that are more realistic with respect to threats posed by an opponent.

Keywords military aircraft routing, network optimization, A* algorithm

Contents

Chapter 1

Introduction

Careful ahead of time planning of military air operations is vital to their success. In addition to maximizing the probability of success, one is interested in saving resources for further operations. Finding the flight path of an aircraft optimized for these objectives simultaneously forms a multi-objective network optimization problem (e.g., [1]) that can be solved with a network search algorithm.

The optimization of flight paths in the context of military air operations is a widely studied subject [2][3][4]. Several network search algorithms are compared in [3] and the conclusion is that a properly formulated A* network search algorithm dominates all known algorithms when comparing their time and memory requirements. A parallel A* algorithm to solve a similar problem has also been implemented in [3].

Previous work in military aircraft routing was also done by H. Puustinen [1]. His work describes an application that finds efficient flight paths by utilizing Dijkstra's shortest path algorithm [5]. The paths are optimized for distance, fuel consumption and opponent threats simultaneously. However, the used network search algorithm does not take time into consideration leading to inaccurate predictions of hostile movement. Thus, it would be useful to have edges' weights depend on time. In addition, while Dijkstra's algorithm is a powerful and a widely used network search algorithm, the A* algorithm often outperforms it by searching fewer nodes and by making heuristic guesses about the shortest path.

This thesis describes a particular route planning algorithm that utilizes the A* network search algorithm to find routes that avoid potential time-dependent threats while minimizing fuel consumption. The model in [1] is extended in several ways. The network building process is made implicit, allowing ad-hoc changes to edge weights and skipping of many edges altogether. Dijkstra's algorithm is replaced with the A* network search algorithm that generally

explores a smaller subset of the network with the help of a problem-specific heuristic function. A temporal dimension is introduced to the algorithm for a more accurate modeling of time-dependent threats. With the edges of the network depending on time a new node dominance relation is introduced to keep the size of the search space within practical limits. A few heuristic functions that affect the run time requirements and optimality of the solution of the A* algorithm are also compared to results given by the Dijkstra's algorithm and the original formulation of the problem.

The thesis is organized as follows. Chapter 2 describes the aircraft routing problem considered in this thesis in more details. Both Dijkstra's algorithm and the A* algorithm are defined in Chapter 3. Chapter 4 describes a few heuristic functions and how to apply the modified A* algorithm to the problem with the help of the node dominance relation. Some numerical examples are shown in Chapter 5 before arriving at the conclusions in Chapter 6.

Chapter 2

The aircraft routing problem

The overall goal is to find an aircraft flight path from an aircraft base to one of many projectile launch points and back. The chosen path should be optimized to avoid any air-to-air or surface-to-air threats while minimizing fuel consumption and path length.

The path begins and ends at a friendly aircraft base. The problem description also assumes perfect knowledge of surface-to-air threats and air-to-air threats posed by hostile aircraft bases. Both types of threats define zones that incur a penalty when flying through them. The hostile bases inflict air-to-air threat, which is a cylindrical shape centered around each base. Surface-to-air sites also have a threat-inflicting zone which is defined in more detail in [1].

As the aircraft flies, hostile aircraft also move. The air-to-air threats are modeled as cylinders the radii of which grow linearly as a function of time after the friendly aircraft crosses a defined reaction surface.

The optimal path for the aircraft is one that minimizes a weighted sum of the following four criteria:

- Total path length
- Fuel consumption
- Path length under surface-to-air threat
- Path length under air-to-air threat

The original formulation of the problem is given in more details in [1].

Chapter 3

Algorithm Description

This chapter first introduces Dijkstra's shortest path algorithm [5] and then shows how the A* algorithm [6] is an extension of it.

3.1 Dijkstra's algorithm

Dijkstra's algorithm was described by Edsger Dijkstra in 1959 [5]. It is a popular network search algorithm that finds the shortest path between two nodes in a network with symmetric and nonnegative edge weights.

A network is defined as a set of nodes N and a set of edges E. Each edge connects two nodes $i, j \in N$ with a weight $w_{ij} \in \mathbb{R}$. A network has symmetric edges if it satisfies $w_{ij} = w_{ji}$ for all nodes i, j. Figure 3.1 shows such a network.

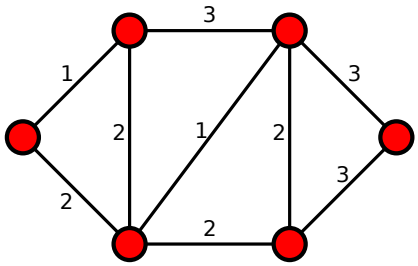

Figure 3.1: A weighted and symmetric network.

The set of edges that connects two nodes $i, j \in N$ via possible intermediary nodes forms a path $P(i, j)$. Additionally, if the path minimizes the sum of edge weights connecting the two nodes, it is marked $P^*(i, j)$ and called the optimal path between i and j. The length of the path is $|P^*(i, j)| = \sum_{\text{edge} \in P^*(i,j)} w_{\text{edge}}$.

The algorithm finds the optimal path between two given nodes. The search begins at a starting node $s \in N$ and finishes when the end node $e \in N$ is found. The output of the algorithm is the path $P^*(s, e)$.

As the algorithm takes steps towards the end node, it keeps track of visited nodes in a set CLOSED. New nodes are inserted into this set when their shortest path to the starting node is discovered. This cumulative path cost is marked as g_i for any node i. The algorithm guarantees that $g_i = |P^*(s, i)|$. Initially only the starting node inserted into CLOSED and $g_s = 0$.

The algorithm proceeds by repeatedly selecting edges that connect node $a \in$ CLOSED to node $b \notin$ CLOSED so that the sum $g_a + w_{ab}$ is minimized. That is, the sum of the length of the path from s to a and the length of the edge w_{ab}. In other words the edge to a non-closed node that yields the lowest cumulative path cost g_b is selected.

Next, the node b is moved to CLOSED and g_b is set to $g_a + w_{ab}$. One can think of CLOSED as a solved set of nodes that expands uniformly away from the starting node. The algorithm terminates when no edge w_{ab} exists or when $b = e$, where e is the preselected end node.

The first iteration of the algorithm is shown in Figures 3.2 and 3.3.

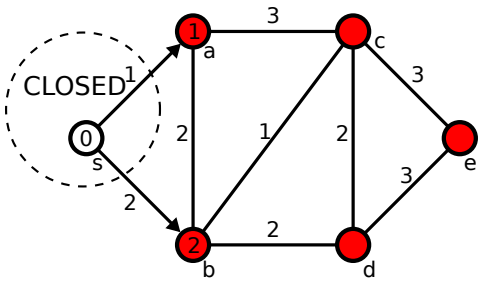

Figure 3.2: A network with nodes a, b, c, d, e and s. The cumulative path cost is marked inside each node. The starting node s is the only member of CLOSED and its neighboring edges to nodes a and b are being inspected. The next step is to take the path to node a, since $g_a = g_s + w_{sa} = 1$ is less than $g_b = g_s + w_{sb} = 2$.

3.2 A* algorithm

The A* algorithm [6] is an extension of Dijkstra's shortest path algorithm. It uses a heuristic function to guide the search in order to avoid visiting nodes that are unlikely to be a part of the optimal path. Whereas Dijkstra's algorithm uses the length of the shortest path from the beginning, $g_i, i \in N$,

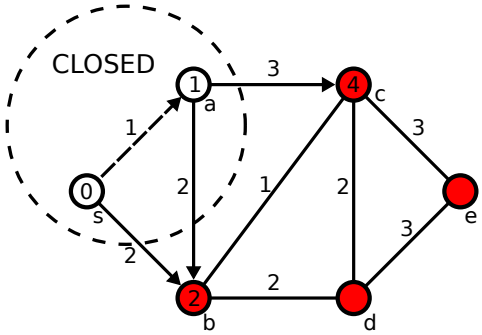

Figure 3.3: Dijkstra's algorithm after one iteration. The nodes s and a are now both in CLOSED. The next step is to compare the cumulative costs of paths $a \to c$, $a \to b$ and $s \to b$: $g_c = g_a + w_{ac} = 4$, $g_b = g_a + w_{ab} = 3$, $g_b = g_s + w_{sb} = 2$. As selecting the path $s \to b$ yields the lowest cumulative cost the node b is added to CLOSED next. The shortest path $P^*(s, b)$ is to travel the edge $s \to b$.

to select edges the A* instead compares $f_i = g_i + h(i)$, where f_i is a heuristic approximation of the length of the whole path and $h(i)$ is the heuristic function.

The heuristic function $h(i), i \in N$, approximates the length of the remaining path from any node i to the end node e. It often takes into account additional information about the properties of the node, for instance, its physical location. The function has to satisfy certain conditions for the solution of the A* algorithm to be optimal and equivalent to the solution given by Dijkstra's algorithm.

The heuristic function shall never overestimate the length of the remaining path. This condition is known as the admissibility condition.

An admissible heuristic function guarantees that $h(i)$ for the remaining path is lower or equal to the length of the actual optimal path, or equivalently,

$$h(i) \leq |P^*(i, e)| \quad \forall i \in \mathrm{N}, \tag{3.1}$$

where $|P^*(i, e)|$ denotes the length of the optimal path from node i to the end node e. The interpretation is that if the remaining path length for a node is overestimated, the algorithm might find a worse path instead.

The heuristic is said to be monotone (or consistent) if it satisfies the condition

$$h(i) \leq h(j) + w_{ij} \quad \forall i, j \in \mathrm{N}. \tag{3.2}$$

A monotone heuristic function is also admissible, since for any optimal path

$P^*(i, e)$ it holds that

$$\begin{aligned}
h(i) &\leq h(j) + w_{ij} && \text{Eq. (3.2)} \\
h(i) - h(j) &\leq w_{ij} \\
h(i) - h(j) + h(j) - h(k) &\leq w_{ij} + w_{jk} \\
h(i) - h(j) + h(j) - h(k) + h(k) - h(l) &\leq w_{ij} + w_{jk} + w_{kl} \\
h(i) - h(e) &\leq w_{i\ldots} + \cdots + w_{\ldots e} && \text{since } h(e) = 0 \\
h(i) &\leq |P^*(i, e)| && \text{Eq. (3.1).}
\end{aligned}$$

If the weights of a network represent physical distances, the heuristic is often chosen to be the euclidean distance between the nodes, which is always shorter or equal to the length of the remaining path.

Chapter 4

Solving the aircraft routing problem with the modified A* algorithm

To formalize the problem defined in Chapter 2 for the A* network search algorithm, the mission airspace is discretized to a three-dimensional grid where neighboring nodes are connected with weighted edges.

The weight of each edge in the network is a weighted sum of the physical length, fuel consumption and threat costs. Fuel consumption is approximated by data from real fuel consumption measurements and it takes into account the aircraft altitude and angle of ascent. The threat criteria are the physical distance flown under air-to-air and surface-to-air threat zones. These are defined in more detail in [1].

The air-to-air threat zone is a high cylinder with a radius that grows linearly in time. The radius grows at the same velocity as a hostile aircraft would fly, which simulates all possible locations for it. The radius starts to grow when the aircraft crosses a reaction surface, that approximates the position after which the aircraft is detected. Figure 4.1 shows how the air-to-air threats influence the weights. Surface-to-air threats each have a separate, time-constant launch acceptability region that defines their threat zone and is defined in [1].

Whereas in the original formulation of the problem [1] all the weights of the network are calculated in the beginning of the search (explicit), the modified A* presented in this thesis calculates them as they are explored (implicit). While this is useful in avoiding edges that are never reached, it is also the only way to calculate them in time-dependent cases.

The following subchapters first show what kind of heuristic functions are cho-

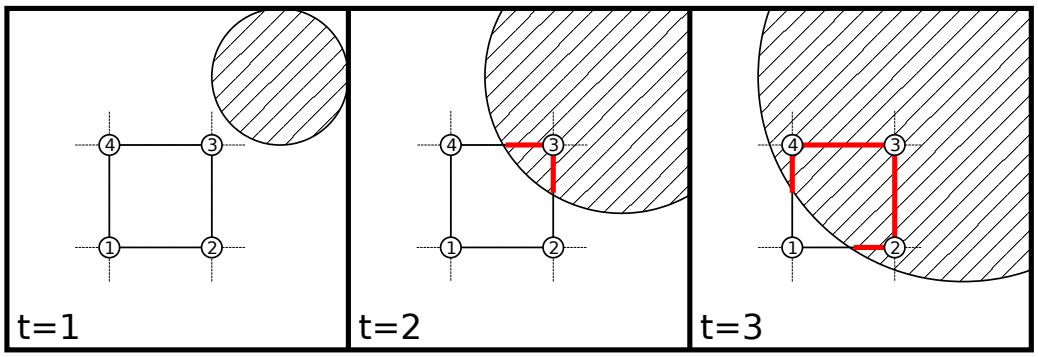

Figure 4.1: The effect of the air-to-air threats to edge costs. The different panels indicate time progression. Edges' air-to-air threat criteria increase in proportion to the covered area.

sen for this problem and then how the algorithm deals with time-dependent weights.

4.1 Heuristic functions

Though a trivial (zero) heuristic function satisfies the conditions defined in Chapter 3.2, an accurate heuristic function has the greatest benefit. Since in the problem each edge weight w_{ij} is a weighted sum of four criteria it makes sense to also define the heuristic function as a similarly weighted sum of specialized heuristic functions for each cost type. For simplicity, however, the air-to-air threat cost heuristic is left out. The full heuristic function is then defined as

$$h(i) = \sum_{f \in \{\text{distance,fuel,surface-to-air}\}} c_f h_f(i),$$

where c_f is the weighting multiplier for the fth criterion and $h_f(i)$ is the specialized heuristic function for that particular criterion.

4.1.1 Distance heuristic

The distance cost is simply the euclidean distance between two adjacent nodes. One of the simplest suitable heuristics is to use the euclidean distance between current node i and the end node e, since by the triangle equality it never overestimates the remaining distance. However, if the network has impassable nodes, this kind of heuristic tries the most direct route first.

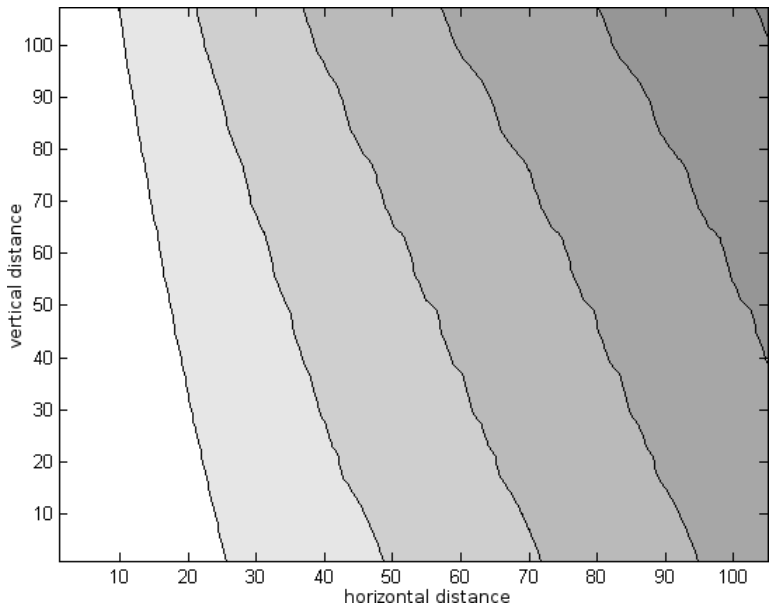

Figure 4.2: A cumulative fuel consumption matrix. Each shade represents fuel costs required to travel to the origin at the lower left corner. The matrix is built from real fuel consumption measurements using Dijkstra's algorithm.

Nevertheless, it remains a widely used heuristic in the literature due to its simplicity.

4.1.2 Fuel heuristic

The fuel consumption component of each weight of the network is based on real fuel efficiency measurements. The A* algorithm can use this knowledge to minimize the total fuel consumption of the mission. Fuel consumption varies as a function of altitude and angle of ascent, so that when paths are optimized only for fuel consumption the aircraft will prefer flying longer distances at high altitudes.

To make heuristic guesses, the algorithm has a table of cumulative fuel consumptions when traveling a certain distance with a given starting altitude. One such table is shown in Figure 4.2.

4.1.3 Surface-to-air threat heuristic

The surface-to-air threat cost is non-zero when the aircraft flies within the threat zone of a hostile surface-to-air site and zero otherwise. Since the threat does not depend on time, it is sufficient to only calculate the minimal

cumulative amount of threat required to escape the zone as the cost is zero everywhere else.

4.2 Treatment of time-dependent weights

This subchapter shows how the A* algorithm is modified to model the time-varying threats in the problem.

In order to describe the time-varying air-to-air threats, the previous definitions in Chapter 3 are extended to include time as a parameter. In particular, edge weights now depend on time, $w_{ij} = w_{ij}(t)$. A useful property based on the problem description in the beginning of Chapter 4 is that

$$\frac{\partial w_{ij}(t)}{\partial t} \geq 0 \quad \forall i, j \in N, \tag{4.1}$$

since only the air-to-air threat criterion depends on time and it cannot decrease as a function of time. With weights depending on time, multiple nodes of the network are also let to exist in the same physical location.

The aircraft is assumed to fly at a constant Mach number corresponding to lower velocities at higher altitudes. Thus, it is possible to calculate the time it takes to fly along an edge from node i to node j simply by $t =$ distance/average velocity. The modified algorithm keeps track of time by marking node arrival times as $t_i, i \in N$. Multiple nodes of the network can now exist in the same physical location with different arrival times. The size of the network is no longer finite, since traveling in circles one can never reach the same nodes again.

These new definitions lead to the unfortunate effect of rapidly expanding the explorable node space unless one manages to eliminate unnecessary nodes. When multiple paths to a new node are explored in the original A*, it suffices to compare the cumulative path length (g_i, the path length traveled so far) and discard all but the shortest path. This works on the assumption that the cost of the remaining path to the end node doesn't change as a function of the path taken so far. However, in a time dynamic case, this assumption no longer holds. A new way to eliminate nodes is required to avoid the exponential growth of candidate paths.

Fortunately, according to the equation 4.1 edge weights are monotone with respect to time, i.e., it is never beneficial to wait or take a path that takes longer in time when cumulative path costs are equal, as shown in Figure 4.1. Similarly, for two nodes $i, j \in N$ with $t_i = t_g$, the consequent path is identical, since the remaining path is only a function of t, and therefore the

other node can be eliminated by comparison $g_i < g_j$. Thus, it is possible to define a dominance relation for two nodes physically in the same location: Node a dominates node b if its time (t_a) and cumulative cost (g_a) are lower or equal to b's and at least one of them is lower. This relation is shown in Figure 4.3.

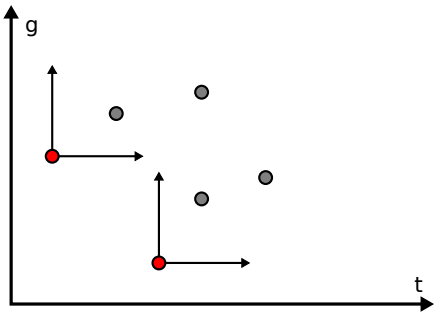

Figure 4.3: The node dominance relation. When a physical location is reached via multiple paths, it is possible to discard many of them merely by comparing their cumulative path length (g_i) and time of arrival (t_i).

This elimination step is important in keeping the time requirements within practical limits. It is worthwhile to note, that if one ignores the time dimension altogether by setting the arrival time t_i of each node to zero, the comparison resembles that of the regular Dijkstra's algorithm where one chooses only the shortest known path to each node.

Chapter 5

Examples

This chapter begins with a comparison of several heuristic functions and then shows example scenarios where the solutions given by the modified A* algorithm differ from solutions given by the original formulation of the problem.

5.1 Comparison of heuristic functions

As long as the chosen heuristic function is admissible and monotone, the result of the A* network search is always optimal. The choice of the heuristic function, however, does affect the number of nodes the algorithm has to search.

The combined heuristic function defined in Chapter 4.1 is compared to two simpler functions: a simple straight-line distance equivalent to the function presented in Chapter 4.1.1 and a constant zero function. For evaluating these three functions, six test scenarios were created. The scenarios differ in the size of the network and in the positioning and amount of surface-to-air threats. Since none of the heuristic functions take time into account, no air-to-air threats were placed in any of the scenarios. The heuristics used the same criteria weights as the edges of the network and they are shown in the Table 5.1. The number of nodes explored in each scenario is shown in Figure 5.1.

Out of the three heuristics tested the full heuristic was able to estimate the weights of the network the best. In all of the six test scenarios, it always found the optimal path with the fewest search nodes. It also made the searches fastest as the time it takes to calculate the heuristic for the nodes themselves is negligible in comparison to the savings made by skipping many nodes during the search.

ID	distance	fuel	surface-to-air
1	0.1	0.01	1
2	0.1	0.01	1
3	1	1	1
4	1	1	1
5	1	0	1
6	0	1	1

Table 5.1: Edge criteria weights for each test scenario. No air-to-air threats were present in the scenarios.

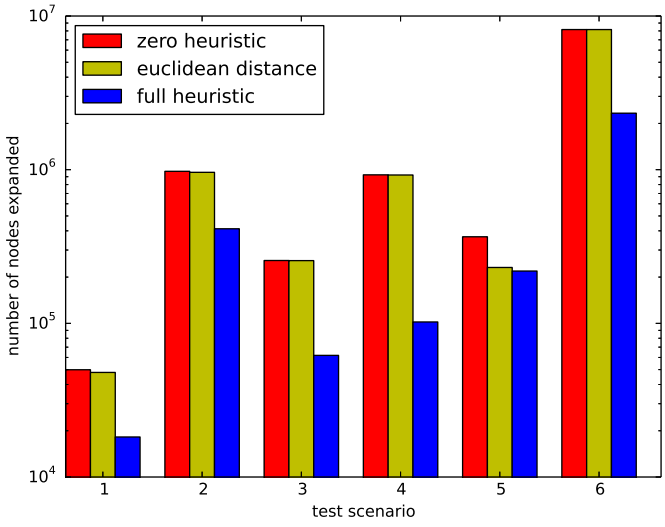

Figure 5.1: The number of network nodes searched by the A* algorithm in six test scenarios for three heuristic functions. Note that the Y axis has a logarithmic scale. Here, the zero heuristic represents a trivial heuristic $h(i) = 0$ for every node i and makes the search equivalent to Dijkstra's algorithm. The euclidean distance heuristic estimates the remaining path cost via a straight-line distance. The full heuristic is a weighted sum of three heuristics and is defined in Chapter 4.1.

5.2 Time-dependent cases

When all of the network edge weights are fixed in time, the solution paths given by the modified A* algorithm are equal to the paths given by the original algorithm in [1]. The results differ when the cylinder-shaped air-to-air threat zones are set to grow in time, allowing the aircraft to fly closer to them without a cost in the beginning of the mission. The threat then has a greater impact on the shape of the return path. This phenomenon is shown in Figure 5.2.

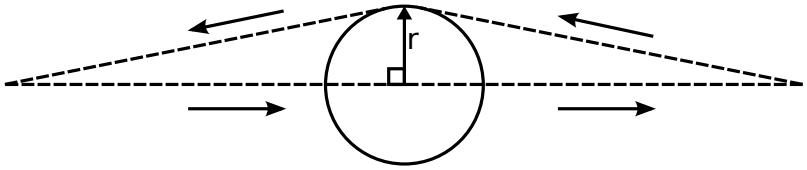

Figure 5.2: The top-down view of a possible flight path with the modified A* algorithm. The path follows the dashed line and begins at the left side of the figure. The circle represents an cylinder-shaped air-to-air threat zone with a growing radius. The return route has to curve around it to avoid air-to-air threat accumulation.

The solutions of the two algorithms also differ when large costs later in the mission can be avoided by saving time earlier on in the mission by flying at a lower altitude. The flight paths in the following example scenario were calculated with both the original algorithm and with the modified A* algorithm. The edges of the network were weighted so that air-to-air threat is avoided as much as possible at the expense of fuel and distance costs. The scenario forces the aircraft to fly briefly through an air-to-air threat.

Following the numbered points inserted into the two solutions in Figure 5.3, the aircraft gain altitude near point **1** to save fuel as the efficiency grows at lower atmospheric pressures. This climbing takes some time, which does not matter as the hostile aircraft only start moving after the reaction surface is crossed as defined in the problem description in Chapter 2.

The aircraft drop to a lower altitude near point **2** for two reasons: due to the shape of the reaction surface, it can travel further before being detected and upon crossing the surface it will have the highest velocity as they maintain a constant Mach number. After **2** the paths diverge: the original algorithm guides the aircraft higher to maximize fuel savings, whereas the modified A* algorithm keeps the aircraft low where it flies faster.

Closing in on the projectile launch point at **3** the bottom aircraft still flies as low as possible. At point **4** it temporarily enters the hostile air-to-air threat zone that has by now grown larger. Having cleared it, the algorithm

no longer has to optimize for speed and by point **5** it has gained the optimal flight altitude for fuel savings. Indeed, it shows that between points **2** and **4** the bottom path is being optimized for flight time and otherwise for fuel. Without the time savings, the penalty for staying inside the air-to-air threat zone (point **4**) would have been larger. The original algorithm follows a fuel-optimal arc between points **3** and **5**. In this scenario, the flight path calculated by the original algorithm is more fuel efficient, but spends more time flying inside the air-to-air threat zone than the solution found by the modified A* algorithm. The modified A* achieves a lower total cost for the path.

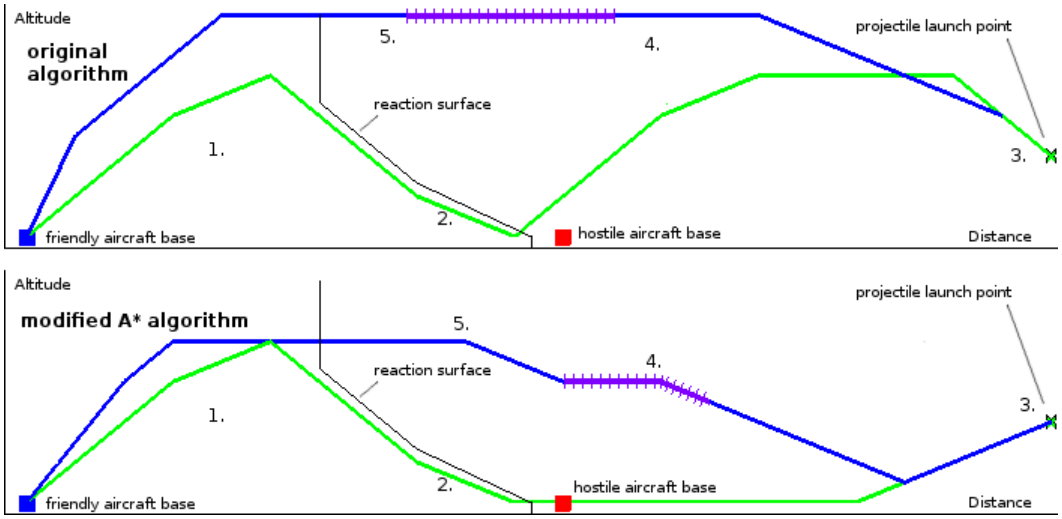

Figure 5.3: The side profile view of a flight path calculated with the original algorith (top) and with the modified A* algorithm (bottom). The optimal path follows the direction given by points **1**...**5**. The connected chain of line segments in the center is the reaction surface. The red square is a hostile aircraft base and the blue square is the friendly aircraft base.

Chapter 6

Conclusions

This thesis described a military aircraft routing problem and a way to solve it with a modified A* path finding algorithm. A node dominance relation was introduced to make the time-dependent problem solvable in practice. The thesis also showed and compared several heuristic functions to be used with the modified A* algorithm.

By approximating hostile movement over time more precisely, the modified A* algorithm produces flight paths that are more useful in practice than the solutions given by the original formulation of the problem. Out of the three heuristic functions tested the full, combined heuristic was able to arrive at a solution with the fewest search nodes.

The node dominance relation was very useful in removing suboptimal search nodes during the search. However, the number of nodes to search still grows quickly when the problem size is increased. In the future, the node dominance relation could be enhanced to also drop nodes that lead to very similar existing solutions or by making further mission-specific assumptions about the relationship between travel time and cumulative path cost.

The full heuristic function was able to estimate the weights of the network the best. In the future, the heuristic function could be extended to also take air-to-air threats into consideration, further decreasing the number of nodes the algorithm has to search.

Chapter 7

Bibliography

[1] H. Puustinen, *Military Aircraft Routing with Multi-Objective Network Optimization and Simulation*, Master's Thesis, Systems Analysis Laboratory, Aalto University, 2013.

[2] M. S. Gudaitis, *Multicriteria Mission Route Planning Using a Parallel A* Search*, Master's Thesis, School of Engineering and Management, Air Force Institute of Technology, OH, 1994.

[3] E. Sezer, *Mission Route Planning With Multiple Aircraft & Targets Using Parallel A* Algorithm*, Master's Thesis, School of Engineering and Management, Air Force Institute of Technology, OH, 2000.

[4] W. M. Carlyle, J. O. Royset and R. K. Wood, *Routing Military Aircraft with a Constrained Shortest-Path Algorithm*, Military Operations Research, Volume 14, Number 3, pages 31-52, September 2009.

[5] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, Volume 1, Issue 1, pages 269-271, June 1959.

[6] P.E. Hart, N.J. Nilsson, B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, Systems Science and Cybernetics, Volume 4, Number 2, pages 100-107, July 1968.

Appendix A

Time complexity of the algorithm

Both Dijkstra's algorithm and the A* algorithm need to repeatedly find nodes that have the lowest cumulative path length. A naive implementation loops through all elements of a set of nodes each time a new node is needed, and new nodes are simply inserted at the end. The computational complexity of the algorithm when using such a linear search is $O(|N|^2)$, where $|N|$ is the number of nodes. Replacing the linear search with a priority queue data structure yields the complexity $O(|E|+|N|\log|N|)$, where $|N|$ is the number of nodes and $|E|$ is the number of edges in the network. Additionally, both algorithms repeatedly check whether a particular node exists in a set. The priority queue is not suited for this task, but an auxiliary hash table data structure can be used to make the check a constant complexity operation.

Appendix B

Yhteenveto (In Finnish)

Sotilaslentokoneiden lentoreittien tarkalla suunnittelulla voidaan vaikuttaa merkittävästi lentotehtävien onnistumiseen. Reitit suunnitellaan kiertämään mahdolliset vihollisuhat kaukaa niin, että säästetään mahdollisimman paljon polttoainetta. Reittisuunnittelu on monitavoiteoptimointiongelma, jota tässä työssä lähestyttiin verkko-optimointitehtävänä.

Lentoreittioptimointia on tutkittu laajasti. Monet olemassa olevat menetelmät perustuvat Dijkstran algoritmiin, jolla voidaan löytää lyhin reitti verkossa kahden solmun välillä. Dijkstran algoritmin soveltamiseksi on verkko määriteltävä niin, että vihollisuhat ja polttoaineenkulutukset tietyssä pisteessä kasvattavat sitä vastaavan kaaren painoarvoa. Tällöin lyhin reitti verkossa on sellainen reitti, joka minimoi polun varrella kuljettavien optimointikriteerien summan. Diplomityössään Heikki Puustinen määrittelee tavan ratkaista lyhimpiä lentoreittejä Dijkstran algoritmilla verkolle, jonka kaarien painoarvot on laskettu etukäteen.

Kirjallisuudessa toinen laajasti käytetty reitinhakualgoritmi on A*-algoritmi. Se laajentaa Dijkstran algoritmia lisäämällä siihen niin kutsutun heuristiikkafunktion, jolla hakua ohjataan kohti tavoitesolmua. Perusmuodossaan A* vähentää tutkittavien solmujen määrää merkittävästi, mutta ratkaisee silti optimaaliset reitit.

Todellisuudessa vihollisuhat voivat liikkua. Olisi hyödyllistä, jos mallista saisi ratkaisuksi reittejä, jotka ennakoivat vihollisliikkeitä eri aikoina. Eräs tällainen liike on esimerkiksi vihollislentokoneen eteneminen, jonka arvioimiseksi tarvitaan sijainnin lisäksi kellonaika. Kun verkon kaarien painoja tarkastellaan eri kellonaikoina, ei verkkoa voida määrittää etukäteen vaan sitä on tarkasteltava reitinhaun edetessä.

Dijkstran algoritmi ei perusmuodossaan sovellu tällaisen verkon ratkaisemiseen. Kandidaatintyössä laajennettiin Puustisen diplomityön reittioptimoin-

timallia lisäämällä siihen muokattu A*-algoritmi, joka huomioi ajan etenemisen polussa ja täten pystyy mallintamaan vihollisuhkien liikettä lentokoneen edetessä.

Kun verkon kaarien painot saavat eri arvoja eri aikoihin, voidaan ajatella, että solmuihin on mahdollista saapua useita kertoja eri aikoihin. Tämä johtaa verkon solmujen määrän eksponentiaaliseen kasvuun haun edetessä. Tarvitaan keino solmujen määrän rajoittamiseksi, jotta reitin voi löytää lyhyessä ajassa. Tässä kandidaatintyössä esiteltiin dominanssirelaatio, jolla voidaan vähentää tutkittavien solmujen määrää merkittävästi.

Ongelmassa sotilaslentokoneelle haetaan reitti lentotukikohdasta aseen laukaisupisteelle ja takaisin tukikohtaan. Lisäksi verkkoon on määritelty solmuja, jotka aiheuttavat vihollisuhkia ympärilleen: ohjuslavettisolmut aiheuttavat uhan, jonka alueella sijaitsevat verkon kaaret saavat suuremmat painoarvot. Kun lentokone lentää erikseen määritellyn reaktiotasan läpi, oletetaan, että vihollisjoukot lähettävät vastajoukkoja lentotukikohdistaan. Nämä liikkuvat vihollisuhat mallinnetaan sylintereinä, joiden säde kasvaa vakionopeudella ajan edetessä ja jotka aiheuttavat vastaavan kasvun verkon kaarien painoihin.

Jokainen kaari verkossa saa painoarvon, joka on painotettu summa neljästä kriteeristä: kaaren pituudesta, polttoaineenkulutuksesta kaarta pitkin kuljettaessa, ohjuslavettiuhasta ja ajassa muuttuvasta vihollislentokoneuhasta. Muuttamalla näiden kriteerien keskinäistä painotusta voidaan löytää eri tavoitteille optimoituja reittejä. Jos esimerkiksi halutaan täysin välttää kaikkia vihollisuhkia ja sallia pidempi reitti, voidaan uhkien keskinäistä painotusta muuttaa.

A*-algoritmin keskeinen osa on heuristiikkafunktio $h(x)$, jossa x on jokin verkon solmu. Heuristiikkafunktio palauttaa arvion matkan pituudesta solmusta x kohdesolmuun (kaarien painojen summa). Reitinhaun edetessä suositaan solmuja, joille heuristiikkafunktio antaa pienen arvon. Jotta menetelmän ratkaisut olisivat optimaalisia ja samat kuin Dijkstran algoritmilla saadut, heuristiikkafunktion on oltava luvallinen (eng. admissible). Luvallinen heuristiikkafunktio on sellainen, joka ei koskaan yliarvioi kuljettavan matkan pituutta.

Jotta ongelman voitiin ratkaista A*-algoritmilla, tuli sille laatia sopiva heuristiikkafunktio. Kun verkon kaaret ovat painotettu summa useasta kriteeristä, myös heuristiikkafunktio on painotettu summa usean erikoistuneen funktion arviosta. Arviot laadittiin matkan pituudelle, polttoaineenkulutukselle ja ohjuslavettiuhalle. Liikkuvien vihollisuhkien heuristiikkafunktion toteutus jätettiin jatkokehityskohteeksi. Tältä osin oikea kustannus kohdesolmuun aliarvioidaan.

Matkan pituutta arvioitiin yksinkertaisella euklidisella etäisyydellä kahden solmun välillä. Se on yleisesti käytetty heuristiikka, joka ei koskaan yliarvioi

jäljellä olevan matkan pituutta, sillä matka linnuntietä on aina lyhyempi tai yhtä pitkä kuin oikea reitin pituus.

Polttoaineenkulutuksen arvioimiseksi laadittiin taulukko, josta voitiin lukea arvioitu kulutus, kun tiedetään solmujen korkeudet ja niiden välinen etäisyys. Sen täydentäminen tapahtui Dijkstran algoritmilla, jolla haettiin pienin polttoaineenkulutus lähtösolmusta kaikkiin muihin taulukon solmuihin. Myös ohjuslavettiuhalle laadittiin vastaava tietorakenne, josta voitiin arvioida suurin mahdollinen ohjuslavettiuhka kahden pisteen välillä.

Heuristiikkafunktioiden vertailussa laadittiin kuusi esimerkkitehtävää, joissa uhkien keskinäisiä painoarvoja vaihdeltiin. Vertailun kohteina oli kolme funktiota: Dijkstran algoritmia vastaava nollafunktio, pelkkä euklidinen etäisyys ja täysi kolmeen komponenttiin jaettu heuristiikka. Kaikissa tapauksissa viimeisin löysi ratkaisun selkeästi pienimmällä määrällä hakusolmuja.

Muokatun A*-algoritmin ratkaisemia lentoreittejä verrattiin Dijkstran algoritmin ratkaisuihin. Reitit vastasivat toisiaan tapauksissa, joissa vihollisuhat olivat vakioita ajassa. Ongelmissa, joissa esiintyi liikkuvia vihollisuhkia, löysi muokattu A*-algoritmi ratkaisuja, jotka saattoivat optimoida osia lentoreitistä lentonopeudelle minimoidakseen vihollisuhan myöhemmässä vaiheessa tehtävää.

Työssä esitelty muokattu A*-algoritmi tuotti hyödyllisempiä lentoreittejä sotilaslentokonetehtäviin kuin aiempi Dijkstran hakuun perustuva menetelmä. Se tuottaa reittejä, jotka ennustavat vihollisten liikettä ajassa ja optimoi reitit siten myös lentoajalle. Työssä esitelty verkon solmujen dominanssirelaatio oli tärkeä reitinhaun komponentti, jolla mahdollistettiin ratkaisujen laskeminen kertaluokkia nopeammassa ajassa. Vertailluista heuristiikkafunktioista komponentteihiin jaettu heuristiikkafunktio löysi tulokset pienimmällä määrällä hakusolmuja.