Fast Graph-Search Algorithms for General Aviation Flight Trajectory Generation

Eran Rippel*, Aharon Bar-Gill† and Nahum Shimkin‡

Technion—Israel Institute of Technology, Haifa, Israel

May 24, 2004

Abstract

We consider numerical algorithms for on-board flight trajectory generation and optimization in three dimensional space. Our approach relies on graph search algorithms, which perform a global search over the set of feasible trajectories. We start by formulating a simplified kinematic model that is appropriate for General Aviation aircraft. The cost function to be optimized includes, the cost function to be optimized accounts for position-dependent criteria such as the flight altitude and terrain clearance. Additional "dynamic" components that involve the angular velocities are introduced to account for riding qualities and pilot workload. Using an approximate grid-based discretization scheme, we transform the continuous optimization problem into a search problem over a finite graph, and apply Dijkstra's shortest-path algorithm to this problem. To reduce the computation time to acceptable levels, we introduce a novel state reduction technique that leads to sub-optimal search. Further speedup is achieved by heuristic search techniques and hierarchical methods. Performance of the proposed algorithms is evaluated for a trajectory optimization problem with terrain following over a 100×100 kilometer area. Our experiments demonstrate the potential of these algorithms, when combined, to provide an on-board solution to realistic flight trajectory generation problems.

*Graduate student, Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, Israel
†Department of Aeronautical Engineering, Technion—Israel Institute of Technology, Haifa, Israel
‡Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, Israel

Nomenclature

a_{vertical}, $\bar{a}_v$	–	aircraft vertical acceleration, and its upper bound
$a_{\text{horizontal}}$, $\bar{a}_h$	–	aircraft horizontal acceleration, and its upper bound
$c(X, U)$	–	immediate cost function
$C(v, u)$	–	cost of edge (v, u)
d_0	–	trajectory accuracy
D	–	set of destination vertices
$DTM(t)$	–	ground level from Digital Terrain Map
$\mathcal{E}$	–	the set of edges of a graph
g	–	cost-so-far estimation function
G	–	a graph
h	–	cost-to-go heuristic function
h_0	–	safety clearance height
H_0	–	clearance height
H	–	flight level
h_x, h_y, h_z	–	3D grid spacing
i, j, k	–	integer grid indices
J	–	cost functional
m, n, ℓ	–	integer indices for neighboring grid points
m	–	maximal number of neighbors for each 3D point
n	–	number of 3D points in the mesh
N	–	number of vertices in a graph
P, P_i	–	a point in 3D Cartesian space
$R_{\min}$	–	minimal aircraft turn radius
s	–	source vertex
S	–	set of vertices with known shortest-path values
$\mathcal{S}_0$	–	finite grid in Cartesian space
t_f	–	first arrival time to a goal state
u, v	–	vertices
U	–	control signal
$V(X)$	–	aircraft velocity at state X
$\mathcal{V}$	–	the set of vertices of a graph
w_κ	–	cost criterion weight
x, y, z	–	Cartesian coordinates
X	–	aircraft state
α	–	tradeoff factor
γ	–	vertical flight path angle
κ	–	cost criterion index
ψ	–	heading angle
$\mathcal{X}_f$	–	set of goal states
Ω	–	set of cost criteria
$\Delta\gamma_0$	–	largest separation between discrete angles

1 Introduction

General Aviation (GA) constitutes a considerable part of today's saturated air traffic. In most instances, a GA aircraft is flown by a single pilot, who may find himself under adverse weather conditions, operating in Single Pilot Instrument Flight Rule (SPIFR) mode; a faulty link to Air Traffic Control (ATC) may complicate matters even more. The additional stress and workload cause many unfortunate accidents. Technological advancements, combined with increased affordability, open new venues for tackling this SPIFR safety problem. These include, for example, Ground Collision Avoidance Systems (GCAS) [1, 2] that employ Geographic Information System (GIS) databases [3, 4].

Three-dimensional (3D) flight trajectory generation is a complex task, involving various criteria that affect the resulting path. These include, among others, the flight duration, the degree to which different terrain features are utilized, the workload imposed on the pilot in order to follow the prescribed trajectory, and riding qualities (or passenger comfort) which are affected by the accelerations involved. Every generated trajectory has to accommodate for these often conflicting objectives. Evidently, the specified cost components and the tradeoff between them should reflect the specific requirements of the flight mission. For example, low altitude may be required to avoid flying within a cloud layer, which may add to pilot stress. 'Sunday pilots', in particular, would prefer maximum dwell time below cloud ceiling, i.e., flying as low as possible, subject – of course – to safety constraints. The weight given to the pilot workload (pumping the stick and the pedals) should depend on the pilot's preferences. And minimal exposure to high g-levels may be critical if an injured passenger is carried onboard. All these considerations bring out the need for a flexible flight trajectory generation capability.

In this paper we address the problem of on-board generation and optimization of 3D trajectories. GA applications call for a low-cost solution to this problem that would trace a flight path for rescuing a person, for example, seconds after destination has been decided on (hospital most suitable to treat the particular injury). Thus, by *on-board* algorithms we refer here to algorithms that can compute optimal flight paths within few tens of seconds. Modern technology should allow for such a solution, in particular under circumstances where communication with ATC fails and routine VFR/IFR (Visual/Instrumental Flight Rules) cannot be maintained.

An implementation of such a trajectory generator includes an onboard computer, in which

the Digital Terrain Map (DTM) resides, and an altimeter (e.g., a GPS receiver), on top of the standard navigational instrumentation normally available in GA airplanes. Laptop computers with DTMs are indeed increasingly being used by light aircraft and helicopters who fly rescue and other emergency missions without adhering to prescribed routes. Also, the provisioning of cues that enable the pilot to comfortably follow the planned trajectory is readily implementable even in today's low-cost aircraft. As stated, in the sequel we focus on the trajectory generation task.

Overview of the proposed approach: A control-oriented formulation of the trajectory optimization problem calls for finding a control signal $U(t)$ to minimize the cost functional:

$$J = \int_0^{t_f} c\left(X(t), U(t)\right)\, dt \tag{1}$$

subject to the nonlinear differential state equation:

$$\dot{X}(t) = f\left(X(t), U(t)\right)\,, \qquad 0 \le t \le t_f\,. \tag{2}$$

In addition, various constraints may apply to the state and control variables. Theoretical solution approaches to this problem include Dynamic Programming, which iteratively computes the optimal value function from any state, and the Calculus of Variations (with the associated Euler-Lagrange equations), which traces the optimal path [5, 6, 7]. Solution of practical problems requires numerical methods [8, 5, 9]. State of the art numerical methods related to the Calculus of Variations are essentially based on non-linear optimization algorithms, such as Gradient Descent and Newton's method. These methods are basically of local nature, so that convergence to the global minimum cannot be guaranteed unless the problem is highly structured. This is not the case in the GA flight generation problem, where rugged terrain and forbidden flight zones, for example, may create multiple locally-optimal trajectories. Dynamic Programming, on the other hand, is essentially a global method. However, due to its exhaustive nature, it is considered computationally feasible only for problems with low state dimension and coarse grid discretization (cf. [5], Chapter 7, and [8]).

The present work aims to develop 3D trajectory optimization methods which are of global nature, yet computationally feasible, using the basic tool of graph-search. Graph search methods are a standard tool for shortest-path problems. However, existing work does not address problems with dynamic aspects, namely when accelerations are taken into account as part of the problem constraints and cost. We note that graph search methods are closely related to Dynamic Programming, but employ the special structure of the shortest-path problem to enhance computational efficiency.

To begin with, we set up a simplified model that captures the essential ingredients of the trajectory optimization problem for General Aviation aircraft. We apply to this model an approximate discretization scheme, which allows the application of efficient global search algorithms to find the optimal path. In particular, we discretize the model over a state space which includes both spatial position and velocity, and formulate the trajectory optimization problem as a shortest-path search problem over a finite graph. This form is appropriate for the application of Dijkstra's shortest-path algorithms, which is an efficient *single-pass* algorithm. In essence, Dijkstra's algorithm proceeds from the target (or initial) state and iteratively develops the front of equi-distant states (or graph vertices). Thus, at each step one new node is tagged with its optimal value, until a target state is reached. As will be seen, the sequential structure of this scheme will allow us to introduce some important modifications to the algorithm.

The application of the basic Dijkstra algorithm to the discretized model is still short of providing on-board performance for problems of realistic size. We therefore explore algorithms that accelerate the process of trajectory generation. First, we reduce the dimensionality of the discrete state space by removing the velocity-related components. Nonetheless, the proposed reduced-state algorithm retains the ability to account for the acceleration cost, albeit in a sub-optimal manner. Further enhancements that are considered include the addition of an A^* heuristic, and a hierarchical (coarse-fine) search scheme. The performance of these algorithms is examined and compared in simulation.

Related Research: The nonlinear nature of both the aircraft state dynamics and the performance index J, make the trajectory generation problem hard to solve. We briefly review here some of the solution approaches that have been proposed for this problem. These include definition of various cost function components, different aircraft model formulations, and algorithmic solution variants.

Funk [10] performs vertical-plane optimization only, by means of cubic splines. The splines are generated by solving a nonlinear programming problem. He addresses only the altitude aspect of the cost. Another trajectory optimization solution, utilizing nonlinear programming, is given by Hargraves and Paris [11] – again only for the vertical plane.

Asseo [12] tackles the 3D problem by looking separately at the horizontal and vertical components. His algorithm employs the gradient method, but restricts the gradient search to the horizontal plane only, avoiding the convergence problems associated with vertical constraints. He achieves vertical tracks by fitting parabola segments that clear critical points. This approach involves no vertical optimization. Also, only the kinematics of the aircraft is modelled, not accounting

for aspects of flight dynamics.

Menon et al. [13] use the optimal control approach to obtain trajectories that minimize a linear combination of flight time and Terrain Following (TF) costs. They address the 3D problem by employing the adjoint method [6, 8], and a 1D search. The transformation allows the iterative algorithm to generate 3D trajectories, while optimizing for a single parameter. The algorithm is complex and its computational cost is high. Here too, only aircraft kinematics is considered, not accounting for flight dynamics.

Lu and Pierson [14] use the optimal control approach and address off-line calculation of optimal vertical TF trajectories. They use a point-mass nonlinear dynamic model of the aircraft for their analysis, and utilize the Hamiltonian function, with the inverse-dynamics approach [15].

Spong et al. [16] deal with the problem of bounded inputs, employing Optimal Decision Strategy (ODS), which is a class of pointwise optimal control strategies. Accounting for the bounded nature of nonlinear aircraft dynamics (such as thrust and aerodynamics), Lee et al. [17] apply the ODS method to the TF flight problem. Rehbock et al. [18] convert the ODS method into a linear programming problem, in contrast to the previous work, where it is posed as a quadratic programming problem. Barnard [19] combines the ODS method with the neural networks technique.

Hess and Jung [20, 21] use a Generalized Predictive Control algorithm, that calculates an optimal control sequence several steps ahead. They then apply only the first control of the sequence, and recalculate a new sequence (a 'receding horizon' process). Lu and Pierson [22, 23] also consider predictive algorithms.

Numeric solutions of continuous Dynamic Programming methods are extensively treated in [9]. However, these methods may require multiple passes over the entire state space. For the specific case of the shortest-path problem (which corresponds to a special case of the problem in Equations (1) and (2) with $c(X,U) = c(X)$, $\dot{X} = U$ and $\|U\| \leq 1$), more efficient alternatives are offered by appropriate discretization schemes coupled with efficient graph search procedures. An early example for the use of this approach may be found in [24]. More recently, numerically consistent algorithm (that converge to the exact solution when the spatial resolution is increased) have been proposed in [25] and [26]. All these methods use a Dijkstra-like algorithm to perform a global search over the discretized space. Unfortunately, these algorithms do not accommodate at present the dynamic aspects of our problem, namely the acceleration-related costs and constraints.

The remainder of this paper is organized as follows. The underlying optimization problem, the aircraft model and the cost function are formulated in Section 2. The transformation of the problem into a graph search and the application of Dijkstra's algorithm are described in Section 3. Sub-optimal enhancements are discussed in Section 4. Section 5 presents simulation results that demonstrate the performance of the proposed algorithms. Finally, in Section 6 we conclude and indicate some topics of interest for further research.

2 Problem Definition

The problem we consider is of generating an optimal aerial trajectory between two points – the *source* and the *destination.* The trajectory has to be flyable, and hence attention is paid to the physics of the problem, including the flight model, acceleration constraints, and cost components that reflect passenger and pilot comfort. We proceed to formulate the continuous model, specify some important maneuverability constraints, and finally specify the proposed cost functional.

2.1 Model Formulation

We start with a simplified, kinematic model of an aircraft flying in a three-dimensional airspace. The aircraft is considered as a point in 3D space, and its translational and angular states at time t are defined by the configuration vector $X(t) = (x(t), y(t), z(t), \gamma(t), \psi(t))^T \in G$. This is illustrated in Figure 1. The set G is defined as $\Re^3 \times [\gamma_{\min}, \gamma_{\max}] \times [-\pi, +\pi)$. The coordinates x, y and z are taken relative to an Earth frame of reference, such as WGS-84 (World Geodesic System). The extremal values $\gamma_{\min}$ and $\gamma_{\max}$ are derived from the flight model, detailed in the next subsection.

The flight speed $V(t)$ will be taken as given, and will not be a controlled variable in our model. This is the common situation is GA flight. However, we do allow the speed to be a (static) function of the configuration vector $X(t)$, namely $V(t) = g(X(t))$. In particular, V may depend on the flight altitude, as well on the climb angle $\gamma(t)$. We note that during climbs, even with throttle compensation, the aircraft can lose roughly 20% of its velocity, and during descends can gain up to 20% increased velocity. Wind speed data may also be used to modifying V according to the heading angle $\psi(t)$. We shall not proceed further here with a precise modelling of these effects.

We assume instantaneous control of the angular rates, so that the control signal $U(t)$ coincides

with $(\dot{\gamma}, \dot{\psi})$. The state equations $\dot{X}(t) = f(X(t), U(t))$ may now be written as

$$\begin{aligned}
\dot{x}(t) &= V(t)\cos\gamma(t)\cos\psi(t) \\
\dot{y}(t) &= V(t)\cos\gamma(t)\sin\psi(t) \\
\dot{z}(t) &= V(t)\sin\gamma(t) \\
\dot{\gamma}(t) &= U_1(t) \\
\dot{\psi}(t) &= U_2(t)\,.
\end{aligned} \tag{3}$$

The control and state variables will be further constrained by the flight model, as detailed in the next subsection. Given an initial state X_0 and the goal set $\mathcal{X}_f$, feasible trajectories are continuous solution paths $X(t)$ that start at $X(0) = X_0$ and terminate at $X(t_f) \in \mathcal{X}_f$. We shall be interested in optimal trajectories, that minimize a cost functional to be defined below.

2.2 Flight Model Constraints

Aircraft performance is subject to physical constraints that affect its maneuverability. Additional restrictions may be applied to ensure passenger and pilot comfort. Assuming that upper bounds are specified for the lateral accelerations, we will find it convenient to represent them by corresponding bounds on the angular velocities and the aircraft turn radius.

Under normal flight condition of GA aircraft, decoupling of the motion plane components is justified, resulting in two approximate acceleration measures:

$$a_{\text{vertical}} = V\dot{\gamma}\,, \qquad a_{\text{horizontal}} = V\dot{\psi}\,. \tag{4}$$

These accelerations approximate the two components of the lateral acceleration relative to the aircraft frame. Accordingly, they should not exceed certain levels for comfort. Assuming hard bounds $\bar{a}_v$ and $\bar{a}_h$ on the vertical and horizontal accelerations, respectively, we obtain the following constraints on the angular rates:

$$|\dot{\gamma}| < \frac{\bar{a}_v}{V}\,, \qquad |\dot{\psi}| < \frac{\bar{a}_h}{V}\,. \tag{5}$$

The minimal turn radius is next determined from $V = \dot{\psi}R$:

$$R_{\min} \geq \frac{V}{|\dot{\psi}|_{\max}} = \frac{V^2}{\bar{a}_h}\,. \tag{6}$$

This minimal radius serves as a constraint imposed on aircraft trajectories. Note that these bounds may depend on the aircraft configuration X through the velocity V. It may be convenient to use some nominal value V_0 for V in these equations, although this is not necessary.

The maximal climb angle, based on climb rate capability safety requirements, gives rise the the additional constraint

$$|\gamma| < \gamma_{\max} \,. \tag{7}$$

We note that different values may be assigned to the upper an lower values of γ and $\dot{\gamma}$ if so required. All the constraints above need to be accounted for in the trajectory planning algorithm.

2.3 The Cost Function

We next specify the class of cost functions that we consider. Referring back to equation (1), the cost functional will be of the form

$$J = \sum_{\kappa \in \Omega} w_\kappa J_\kappa \tag{8}$$

where $\Omega = \{\text{time}, \text{altitude}, , \text{riding\_qualities}\}$ is the set of criteria, possibly augmented by others as discussed below. The weights w_κ control the tradeoff between these sometimes conflicting criteria.

We first consider integral costs of the form

$$J_\kappa = \int_0^{t_f} c_\kappa(X(t))\, dt \,, \tag{9}$$

where the running cost function $c_\kappa(X)$ is to be chosen appropriately. We next indicate some possible choices of special interest. Note that a general terminal cost of the form $\phi(X(t_f), t_f)$ may be accommodated as well, but seems to be of limited use in the present context.

The flight duration t_f is an obvious performance measure. It may be expressed as a unit integral over the flight interval:

$$J_{\text{time}} = \int_0^{t_f} 1\, dt \tag{10}$$

Next, the *length* of the trajectory is obtained by letting $c_\kappa(X) = V$ (recall that V is a function of X in our model). More generally, the fuel consumption rate may be approximated as a function $c_{\text{fuel}}(X)$ of the the flight configuration X, leading to the cost component J_{fuel} which evaluates the

total fuel consumption. We note that in a typical application the fuel consumption is approximately proportional to the flight time, and in our experiments we only consider the latter.

An altitude cost components will be of the general form

$$J_{\text{altitude}} = \int_0^{t_f} F(z(t), DTM(t))dt \tag{11}$$

where $z(t)$ is the aircraft altitude at time t, $DTM(t)$ is the ground height at the current (x, y) position, as may be read in the on-board Digital Terrain Map, and F is an appropriate function thereof. For example, in Terrain Following flight, we might require that the aircraft maintains a fixed clearance H_0 above the ground, and penalize for deviations by

$$J_{\text{altitude}} = \int_0^{t_f} |z(t) - DTM(t) - H_0| \, dt \, . \tag{12}$$

Another interesting case is when the flight altitude is required to remain below a certain level H – for example, to keep the aircraft below a cloud layer. An appropriate cost function may take the form

$$J_{\text{altitude}} = \int_0^{t_f} I\{z(t) > H\}dt \, , \tag{13}$$

where $I\{\,\}$ is the indicator function (that equals one if $z(t) > H$, and zero otherwise). In either case it is natural to impose a hard constraint of the form $z(t) \geq DTM(t) + h_0$, where h_0 is a safety clearance. This is easily incorporated in our graph model.

A second class of cost criteria that will be of interest to us concerns the passenger comfort, as affected by the lateral accelerations. We shall use the term *riding qualities* to refer to this criterion. Using (4), the proposed measure of the riding qualities is

$$J_{\text{riding\_qualities}} = \int_0^{t_f} (|\dot{\gamma}(t)| + \alpha|\dot{\psi}(t)|)V(t)dt \, . \tag{14}$$

This criterion quantifies the lateral acceleration components, its values being accumulated over the entire trajectory. These accelerations influence both the pilot and the passengers. The parameter α allows for relative weighting of the two component.

An additional concern for trajectory planning is the pilot workload. During the flight, the pilot is preoccupied with numerous tasks on top of the need to follow the outlined trajectory. Having a

rapidly varying trajectory requires the pilot to constantly focus on the task of following it, paying less attention to other instrument and visual data. Furthermore, the accumulated workload may lead to excessive stress and fatigue, and should therefore be an important consideration during the trajectory construction process, especially for low flight over irregular terrain. To some extent, the riding qualities cost as defined above accounts also for the pilot workload, as it quantifies the total angular change that the trajectory requires. Other measures of pilot workload may include the *number* of required maneuvers, the total time that the flight is non-straight or non-level, and other related quantities. Obviously, the proper choice of cost here requires careful consideration of the human engineering factors involved. We shall therefore rely on the riding qualities cost to regulate the smoothness of the planned trajectory.

3 The Basic Graph-Search Solution

In this section we propose an approximate solution of the trajectory optimization problem that was presented above. We begin by discretizing the continuous model. We then translate the problem into a weighted graph search problem, which may be solved using Dijkstra's shortest path algorithm.

3.1 Discretization

The proposed discretization scheme begins with a uniform discretization of the three-dimensional space of spatial positions. The discretized trajectory is constrained to go through the grid points, connected with straight line segments. The actual trajectory will of course be a smoothed version of the discrete one. The smoothed trajectory may be computed by a post-processing method, or simply left to be determined by the pilot and the guidance filter.

Consider the discrete grid $\{(ih_x, jh_y, kh_z)\} \subset \Re^3$, where the constants h_x, h_y, h_z determine the grid spacing in the respective coordinates, and i, j , k take integer values. A finite grid $\mathcal{S}_0$ is obtained by intersecting this set with the set of feasible spatial points in continuous space. In particular, the definition of $\mathcal{S}_0$ should incorporate all hard constraints concerning forbidden flight zones and altitudes. The horizontal grid spacings h_x and h_y will usually be identical, which we assume henceforth. The choice of these resolution parameters is discussed below.

We now create a mesh over the spatial grid by defining a neighborhood structure. The neighbors

of a point $P = (ih_x, jh_y, kh_z)$ are those grid points which are accessible from it in one step. The simplest definition that is useful for our purpose is given by

$$\text{Neighb}((ih_x, jh_y, kh_z)) = \left\{((i+m)h_x, (j+n)h_y, (k+\ell)h_z) \ : \ m,n \in \{-1,0,+1\}, |\ell| \leq \ell_{\max}^{m,n}\right\} \tag{15}$$

with $n = m = 0$ excluded. Thus, the horizontal plane neighbors are the eight nearest points (Figure 2a). Other possibilities will be discussed below. $\ell_{\max}$ is constrained by the allowed elevation angle $\gamma_{\max}$ (equation 7) and should therefore obey the following bound:

$$h_z \ell_{\max}^{m,n} \leq \tan(\gamma_{\max}) \sqrt{(mh_x)^2 + (nh_y)^2}\,. \tag{16}$$

For simplicity we assume below that $\ell_{\max}^{m,n} = \ell_{\max}$ is independent of m and n.

As the discrete path is restricted to neighboring mesh points, the specified neighborhood structure determines the discrete values that the flight path angles γ and ψ can assume. Given two neighboring grid points $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$, with $P_2 - P_1 = (mh_x, nh_y, \ell h_z)$, their relative path angles are given by:

$$\psi_{1\to 2} = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) = \tan^{-1}\left(\frac{nh_x}{mh_y}\right) \tag{17}$$

$$\gamma_{1\to 2} = \tan^{-1}\left(\frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}}\right) = \tan^{-1}\left(\frac{\ell h_z}{\sqrt{(mh_x)^2 + (nh_y)^2}}\right) \tag{18}$$

(where inverse tangent in the first equation should be interpreted as the four-quadrant inverse tangent, according to the signs of the nominator and denominator in its argument.)

Given the above neighborhood structure and $h_x = h_y$, the heading angle ψ is constrained to multiples of $\pi/4$. A finer angular resolution may be obtained by extending the definition of the neighboring points beyond the nearest grid points, as illustrated in Figure 2b. Naturally, this extended definition will entail an increase in search time.

We next consider a convenient representation for the discretized state vector $X = (x, y, z, \gamma, \psi)$. We shall use the representation $X = (P_1, P2)$, where the grid point $P_1 = (x, y, z)$ provides the current position, and P_2 gives the path angles according to equations (17)-(18). In fact, we shall consider only such pairs (P_1, P_2) where P_2 is a neighbgor of P_1, namely $P_2 \in \text{Neighb}(P_1)$, so that

γ and ψ are restricted to those values that point directly to neighboring points. This leads to the graph structure that we define below.

The choice of the grid resolution is a decisive factor in the complexity and accuracy of the resultant algorithm. We outline briefly some general guidelines for choosing the grid separation parameters. A specific example may be found in Section 5. Starting with the horizontal parameters $h_x = h_y$, their choice should be directly dictated by the required trajectory resolution. In particular, if the trajectory is required to be specified to within an accuracy d_0 in the horizontal plane, we need to choose $\sqrt{2}h_x \leq d_0$. Moreover, to accommodate the allowed turning radius $R_{\min}$, and assuming that we do not allow discrete heading angle changes of more than 90^o, h_x should not be larger than $R_{\min}$. When the accuracy requirement is the dominant one, as may be expected, we will have $h_x << R_{\min}$. In that case special care is required to enforce the turn radius constraint within the algorithm, as discussed in Section 3.4. Finally, the resulting resolution parameter h_x will typically be larger than the available DTM grid separation, and for convenience may be chosen as a multiple thereof.

The vertical spacing h_z is chosen to satisfy several requirements. The obvious one is the required resolution in the flight elevation. Next is the required resolution in the vertical flight path angle (or climb angle). The largest separation between two discrete angles may be seen to equal $\Delta\gamma_0 = \tan^{-1}(h_z/h_x)$ (this is just the first possible angle above zero when flying along horizontal grid lines), so h_z should be chosen so that this value is small enough. Furthermore, h_z must be small enough so that a change in the (discrete) climb angles between adjacent grid points can be done within the allowed range of vertical acceleration. Specifically, suppose that the maximal vertical acceleration $a = \bar{a}_v$ is applied to the aircraft over a distance h_x, which is traversed in time $T = h_x/V$. Then the vertical increment (over a zero acceleration or straight line path) equals $aT^2/2$, so that we must have h_z not more than that:

$$h_z \leq \frac{1}{2}\bar{a}_v \left(\frac{h_x}{V}\right)^2 . \tag{19}$$

Typical this bound should be fairly large, so that h_z can be chosen based on the required vertical and angular resolutions.

3.2 Graph Structure and Costs

We next define a graph structure for the discrete problem. This includes a finite set of vertices $\mathcal{V}$, a set of edges $\mathcal{E}$ connecting pairs of vertices, and a cost for each edge. We shall also refer to $\mathcal{V}$ as the (discrete) state-space of the problem. Two vertices that are connected by an edge are said to be neighboring vertices. Note the distinction that is made here between neighboring vertices and neighboring points, as defined in the previous Subsection.

A vertex v corresponds to a discretized state X as defined above, namely a pair of neighboring points in the cartesian 3D grid $\mathcal{S}_0$:

$$v = (P_1, P_2) \in \mathcal{S}_0 \times \mathcal{S}_0\,, \quad \text{with } P_2 \in \text{Neighb}(P_1)\,. \tag{20}$$

Here $P_1 = (x_1, y_1, z_1)$ is the current location of the aircraft, and $P_2 - P_1$ defines the direction of its velocity vector. Since the aircraft necessarily follows its specified velocity vector, P_2 must be the aircraft's next location on the grid. A neighboring vertex u of a vertex $v = (P_1, P_2)$ is therefore of the form $u = (P_2, P_3) = ((x_2, y_2, z_2), (x_3, y_3, z_3))$, with P_3 a neighbor of P_2 as defined by equation (15). An *edge* in this graph is therefore of the form $((P_1, P_2), (P_2, P3))$, with $P_1 \in \mathcal{S}_0$, $P_2 \in \text{Neighb}(P_1)$, and $P_3 \in \text{Neighb}(P_2)$.

An example of a vertex (P_1, P_2) and some of its neighbors (P_2, P_3) is depicted in Figure 3. Note that not all neighboring vertices may be feasible due to turn radius constraints, and indeed the figure does not depict neighboring vertices that require a turn of 180^o or more. We will consider this constraint separately in Section 3.4, and use it to further restrict the set of edges of the searched graph.

Having defined the neighbors of a vertex, the cost $C(v, u)$ for each edge can now be specified. In accordance with equation (8), this cost will be in the form $C(v, u) = \sum_\kappa w_\kappa C_\kappa(v, u)$. We start with the cost components of the form (9). Recall that each vertex v defines a configuration vector X_v of the form $X_v = (x_v, y_v, z_v, \gamma_v, \psi_v)$. The flight time between vertices v and u may be approximated as the distance divided by the average velocity, namely

$$\Delta t_{vu} = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2 + (z_u - z_v)^2}/\bar{V}_{vu} \tag{21}$$

where $\bar{V}_{vu} = (V(X_v) + V(X_u))/2$. Consequently, for a running cost function $c_\kappa(X)$ we let

$$C_\kappa(v, u) = \frac{c(X_v) + c(X_u)}{2}\Delta t_{vu}\,. \tag{22}$$

In particular, for the time component from equation (10) we obtain

$$C_{\text{time}}(v,u) = \Delta t_{vu} \tag{23}$$

and for the altitude component from equation (11):

$$C_{\text{altitude}}(v,u) = \frac{F(z_v, DTM_v) + F(z_v, DTM_u)}{2} \Delta t_{vu} \tag{24}$$

where $DTM_v = DTM(x_v, y_v)$.

Next, using the notation of Equations (17)-(18) applied to the vertices $v = (P_1, P_2)$ and $u = (P_2, P_3)$, define

$$\Delta\gamma = \gamma_{2\to 3} - \gamma_{1\to 2} \tag{25}$$

$$\Delta\psi = \psi_{2\to 3} - \psi_{1\to 2}\,. \tag{26}$$

The riding qualities cost component (14) is then given by

$$C_{\text{riding\_qualities}}(v,u) = (|\Delta\gamma| + \alpha|\Delta\psi|)\bar{V}_{vu} \tag{27}$$

We have thus defined the graph $G = (\mathcal{V}, \mathcal{E})$ with non-negative edge costs, $C(u,v) \geq 0$. The required solution is the shortest path on this graph, which can be found using standard graph search algorithms. We next describe the basic Dijkstra algorithm as applies to this problem, and then consider certain modifications that are required in order to account for the turn radius constraints.

3.3 Dijkstra's Algorithm

Dijkstra's algorithm [27] provides an efficient solution to the minimal-path graph search problem. Essentially, the algorithm recursively evolves the front of vertices that are closest to the source, until a target vertex is reached. Convergence is therefore guaranteed.

A pseudo-code implementation of the Dijkstra algorithm is shown in Table 1. $G = (\mathcal{V}, \mathcal{E})$ denotes the graph, C is the cost function over the edges $\mathcal{E}$, s is the source and D the set of destination vertices. The algorithm stores for each vertex v an estimate $g(v)$ of the cost-so-far, namely the cost of the shortest path from the source. $g(v)$ is initialized to $+\infty$, except for the source where in is initialized to 0.

The algorithm also maintains a set $S \subset \mathcal{V}$ of vertices whose final shortest-path values have already been determined. This set is initialized with the source vertex, $\{s\}$, and $g(s) = 0$. The algorithm then repeatedly selects (or *retires*) the vertex v in the complement of S which is closest to the source. This is essentially done by considering all vertices u which are one-edge neighbors to some vertex v is S, and picking up the one with the smaller cost estimate $g(v) + C(u, v)$. This selection is effectively handled by managing a priority queue, which maintains the best cost estimates so far for all neighbors of S. For any vertex u retired to S we also save its predecessor v, so that the optimal path may be traced back.

Dijkstra($G = (\mathcal{V}, \mathcal{E}), C, s, d$)

$S = \{s\}$
$PriorityQueue = \{s\}$
$g(s) = 0$
`for all` $v \in \mathcal{V} \backslash S$,
    $g(v) = \infty$
    $predecessor(v) =$ `none`
    $PriorityQueue$.`insert`(v)

`while` $D \cap S = \emptyset$,
    $v = PriorityQueue$.`extract_minimum`()
    `retire` v `to` S
    `for` $u \in$ `neighbors`(v),
        `if` $(g(u) > g(v) + C(v, u))$ `then`
            $PriorityQueue$.`decrease_key`$(u, g(v) + C(v, u))$
            $PriorityQueue$.`predecessor`$(u) = v$

`Backtrack from` D `to` s

Table 1: Dijkstra's algorithm

It is readily verified that Dijkstra's algorithm does compute the shortest path, under the condition that the link costs are non-negative [28]. The time complexity of the algorithm is $O(N \log N)$, where N is the number of vertices in $\mathcal{V}$, provided that the degree of the graph (number of neighbors for each node) is bounded, and the queue is implemented with an efficient data structure. To realize that, note that the algorithm performs a single-pass over the state space, with each vertex requiring at most one priority queue extraction and a number of priority queue updates bounded by the number of its neighbors. An efficient priority queue data structure, such as the binary heap,

can handle extraction of a minimal element or value update in a list of N elements at $O(\log N)$ each.

Returning to the flight trajectory optimization, Dijkstra'a algorithm can be directly applied to solve this problem, provided that we ignore for the moment possible turn radius constraints. To estimate the time complexity of the algorithm, assume that the 3D mesh has n points. Let m be the maximal number of neighbors for each point. For the neighborhood structure of Equation (15), we have that $m \leq 8(2\ell_{\max} + 1)$. As a vertex in a pair of two neighboring points, the number of vertices is $N \leq mn$, and the basic Dijkstra algorithm runs at $O(N \log N)$.

3.4 Constrained Dijkstra

An important factor that has been left out of the above algorithm is the turn radius constraint. In a typical situation, the minimal turn radius in the horizontal plane (as per Equation 6) may be significantly larger than the horizontal resolution h_x. In that case, in order to allow large single-step heading changes as required in our scheme, the turn radius constraint should be enforced by considering a span of several consecutive vertices. For example, following a 90^o turn, the prescribed trajectory should not have another heading change (in the same direction) for a certain stretch of flight, so that the aircraft can complete the first maneuver.

An optimal solution to such multi-step constraints would require to expand the state space into sequences of point of required length. This would easily leads to an increase of the state space by orders of magnitude, with a corresponding increase in the computation times. To avoid this problem, we propose a *sub-optimal* modification of the basic Dijkstra algorithm, which retains the original state space.

The *Constrained* Dijkstra algorithm utilizes the basic Dijkstra data structure, which allows to trace back the optimal path (to a required number of steps) for each retired vertex. Recall that whenever a state is retired (to the set S), it causes an update in the estimated cost-so-far of its (non-retired) neighbors. In the proposed modification, when a vertex is retired, it traces back the recorded path to the required number of steps, and then determines the set of its feasible neighbors in light of the added multi-step constraints. Thus, the trajectory cannot proceed to vertices that violate these constraints. Other than that, the algorithm proceeds exactly as before. It is evident that the a complexity of the algorithm remains the same, namely $O(N \log N)$.

To handle the turn radius constraint, it remains to specify the sequences of vertices that obey or violate this constraint. This can be done in different ways, and we specify below a simple heuristic criterion that will also be used in our experiments. With each (discrete) change in the heading angle of magnitude α, we associate a distance d before and after the turn point. d designates the travel distance required to complete the turn under the turn radius constraint $R_{\min}$. Using the geometry in Figure 4, we obtain $d = R_{\min} \tan(\theta/2)$. This rule can be easily converted into a "dictionary" of allowed turn sequences over the discrete grid.

As an illustration for the effect of the turn radius constraint, we depict in Figure 5 two trajectories. One is created by the basic Dijkstra algorithm, while the other imposes turn radius constraints via the constrained Dijkstra algorithm. The general conditions of the simulation are as detailed in Section 5. In this example, an altitude cost component penalized the aircraft for an altitude of more than 100 meters above the ground, and the lateral acceleration cost components were removed in order to highlight the influence of the turn radius constraint. The smoothing affect of this constraint is apparent in the form of the resulting trajectory.

4 Accelerating the Algorithm

The previous section described a Dijkstra-based solution to our trajectory optimization problem. Despite the efficient search procedure, running times are still excessive due to the large cardinality of the disceretized state space. For quantification metrics, see Section 5.

In this section we present several approaches for accelerating the Dijkstra search. We begin by describing an $\mathbf{A}^*$ enhanced version, and explain why the acceleration we obtain is only minor.

We focus therefore on two additional Dijkstra-based algorithms that tradeoff accuracy for speed. The *Reduced-state* Dijkstra algorithm cuts down the number of vertices in the search graph, by redefining a vertex as a spatial point without the angular (or next-point) components. Consideration of the angular components is retained through the back-tracking property of the Dijkstra search. The *Hierarchical* Dijkstra algorithm conducts an initial search at a coarser spatial resolution, which constrains the finer search to a smaller area of the map. Each algorithm by itself achieves a significant speedup compared to the basic Dijkstra algorithms of the previous section. Combined, the two modified algorithms offer an on-board solution to the trajectory generation problem. As we demonstrate empirically in Section 5, the performance degradation due to these sub-optimal

modifications is minor.

4.1 Reduced-State Dijkstra

The major factor that slows down the Dijkstra algorithm is the large state space that needs to be explored. As was discussed before, our Dijkstra algorithm employs a state (or vertex) which comprises of two neighboring spatial points, resulting in a state-space cardinality of mn, where n is the size of the spatial grid and m is the average number of neighbors to each point in that grid (see Section 3). This two-point state decodes the flight velocity vector so that we can deduce the angular acceleration (and its associated costs) between two neighboring vertices. The underlying idea behind the *Reduced-State* Dijkstra is to settle for a single-point vertex, while still accounting for all cost components, albeit in a sub-optimal way. To take account of acceleration cost components, we shall use the trace-back feature which is inherent in the Dijkstra algorithm. The basic idea is very similar to the one used in Section 3.4 to enforce the turn-radius constraints.

The Reduced-State Dijkstra algorithm thus reduces the state-space to n vertices. Its running time is therefore $O(n \log n)$. A vertex is defined to be a single point in the 3D mesh, i.e., $v = X = (x_1, y_1, z_1) \in \mathcal{S}_0$. A neighbor of vertex v is defined by equation (15).

Computation of the angular acceleration (and the associated riding qualities cost) requires three consecutive spatial points. Referring back to Dijkstra's algorithm in Table 1, recall that for each retired vertex v (now a single spatial point) the algorithm stores its predecessor vertex in order to trace back the optimal path. Now, whenever a vertex is retired, the algorithm is required to update the cost-to-go estimates for its neighbors. For such a vertex v we use following notation: $\mathrm{pre}(v) = (x_1, y_1, z_1)$ is its predecessor vertex; $v = (x_2, y_2, z_2)$ is the current vertex; and $u = (x_3, y_3, z_3)$ is neighboring vertex. With these definitions, the edge cost $C(v, u)$ can be computed by using exactly the same equations (23)–(27) that were used before. Turn radius constraints may be incorporated into this algorithm exactly as described in Section 3.4.

It should be apparent that the search algorithm proposed here is sub-optimal when all components of the cost function are present. Specifically, it cannot fully account for the riding-quality cost that makes the optimal value function depend also on the directional components of the full state. Instead of exploring all possible velocity vectors (or predecessor points) in a given spatial point, we freeze the velocity vector that leads to it the first time that it is reached. Other options

(which could eventually turn up to be the optimal ones) are suppressed and excluded from further consideration. Nonetheless, the cost discrepancies between the results obtained by full-state the Dijkstra solution and those obtained via the Reduced-State Dijkstra solution turn out to be minor in all our experiments. One way to explain this in the context of our problem is by noting that (small) deviations from the optimal velocity can usually be corrected later on with little penalty. The speedup achieved by the state reduction is, however, most substantial.

4.2 The A^* Algorithm

The A^* algorithm, introduced by Nilsson [29], accelerates graph searching procedures by adding a heuristic cost component to direct the search towards an optimal route. Essentially, the heuristic cost function is an *a-priori* estimate of the cost-to-go function from each vertex. In our problem, the search procedures used so far did not make use of any information concerning the location of the target state. It seems reasonable, for example, to give some priority to those vertices that point in the direction of the goal, rather than those that point in the opposite direction.

Recall that the Dijkstra algorithm used the cost-so-far function $g(v)$ to direct the search. This function is used to choose the vertex that is explored and retired at every step (by determining the order of vertices in the priority queue). The A^* algorithm uses for that purpose a modified cost $\tilde{g}(v)$, where $\tilde{g}(v) = g(v) + h(v)$, and $h(v)$ is a pre-specified heuristic. The algorithm, as used here, is otherwise identical to the Dijkstra algorithm described above. Nilsson [29] proved that when $h(v)$ is an *underestimate* of the minimal cost-to-go between the current vertex v and the destination vertex, then A^* produces an optimal solution. In the ideal case that $h(v)$ equals the optimal cost-to-go, the algorithm directly finds the optimal path. Setting to $h(v) = 0$ recovers the basic Dijkstra algorithm.

The challenge in designing the A^* algorithm is in finding a good underestimating heuristic. We approach this task by examining separately each component of our cost function.

An underestimating time heuristic should be smaller than the (optimal trajectory) flight time from any given vertex v to the destination vertex d. A simple time heuristic is therefore the Euclidian distance from our current position to the destination, divided by the speed. The accuracy of this estimate depends of course on the deviation of the optimal trajectory from the straight line, which depends mainly on the other cost components.

On the other hand, finding a reasonable underestimate for the altitude cost or the riding qualities cost is more challenging. If we look at each of these cost components separately while completely disregarding the others, we obtain an underestimate that tends to zero, which is obviously not helpful. For example, the altitude cost is an indication of how high above the terrain the aircraft is flying – the aircraft can simply fly at the zero-cost height all the way to nullify that cost component. When only the riding qualities are considered, the best path is a direct course towards the destination, also resulting in near-zero cost.

So far, we have not been able to come up with an easily computable heuristic that adequately approximates these cost components in the "interesting" cases (namely, where the optimal flight path is not a straight line). Some initial attempts at analyzing the spatial regularity of the terrain (using Fourier Transforms and Wavelet methods) have not led to noticeable improvement in the algorithm. We therefore restrict further consideration to the above-mentioned straight-line heuristics for the flight time, which will be examined in the experimental section.

4.3 Hierarchical Dijkstra

While the Reduced-State Dijkstra algorithm decreases the total number of vertices in the graph, the Hierarchical Dijkstra restricts the detailed search to a specific segment in the overall graph. Hierarchical and multi-grid schemes are of course widely-used tools in numerical analysis and optimization, and we describe here a simple coarse-fine scheme that is convenient for our application. The search problem is first solved at a coarser level, which is obtained by some down-sampling of the original spatial grid.

The implementation considered here uses a down-sampling by a factor of k along each axis, leading to a k^3 reduction in the spatial grid. The DTM was correspondingly down-sampled. Since safety is a major concern, the ground elevation at each sampled point was taken as the maximal elevation over the map points it covers.

The algorithm calculates an optimal path at the coarse level, using the down-sampled DTM. The resulting coarse-level trajectory defines a search corridor over the original (fine) spatial grid. The points in this corridor are simply those that are within a specified distance from the coarse trajectory. The fine-level search is then conducted inside this corridor, see Figure 6. The same algorithm may be used for both the coarse and fine level searches.

5 Experimental Results

This section describes some simulation experiments that demonstrate and compare the performance of the proposed algorithms. We consider the following issues:

a. Performance of the Reduced-State Dijkstra algorithm, as compared to the standard (full state) algorithm.

b. Effect of the cost parameters on the obtained trajectory.

c. The A^* and Hierarchical enhancements to the reduced-State Dijkstra algorithm.

The DTM that was used is a 100 by 100 km digital map with a 50 m resolution grid. The map coordinates are

bottom-left (origin):	N32^{o}10$'$, E34^{o}50$'$
top-right (destination):	N33^{o}10$'$, E35^{o}50$'$

The state at the origin point is specified by $\psi_0 = 45^o$ (heading towards the destination), $\gamma_0 = 0$. These variables are left free at the destination points. The initial and final heights are set to ground level. We assume a constant flight velocity of $V = 100\,\mathrm{m/s}$, and a required bound of 5 $\mathrm{m/s^2}$ $(g/2)$, on the two lateral acceleration components. The cost function was taken according to Equations (8), (10), (12) and (14). The following weights were used, unless otherwise stated: $w_{\mathrm{time}} = 0.2$, $w_{\mathrm{altitude}} = 1$, and $w_{\mathrm{riding\_qualities}} = 1$. These weights were used after empiric experimentation found them to give a good tradeoff between the various cost criteria.

The basic horizontal resolutions (h_x and h_y) were set to 800 meters. This presents a reasonable trajectory resolution for general aviation. The DTM was accordingly down-sampled. The minimal turn radius constraint from equation (6) computes as $R_{\min} = 2000\,\mathrm{m}$. As this is larger than h_x, the turn radius constraint will be enforced throughout by using the Constrained Dijkstra method of Section 3.4. The vertical resolution parameter h_z was set to 30 meters (100 feet), with $\ell_{\max} = 2$ (see (15)). This gives an angular resolution of $\tan^{-1}(h_z/h_x) = 2.15^o$ in the vertical flight path angle γ, with $\gamma_{\max} = 4.3^o$. The acceleration constraint from equation (19) requires $h_z \leq 160\,\mathrm{m}$, which is well satisfied.

The range of flight elevations considered was 1500 m, yielding 50 points in the vertical axis, and total number of states in the spatial grid is $n = 125^2 \times 50 = 625000$. The number of neighbor points

in $m = 8 \times (2\ell_{\max} + 1) = 40$, which brings the number of vertices in the *full-state* formulation to $N = mn = 25 \times 10^6$. Simulations were conducted on an Intel® Pentium® 3, 866 MHz with 0.75 GB RAM CPU. The algorithmic engine is a Microsoft® Visual C++ program, and the graphics engine is MathWorks MATLAB®.

The first experiment compares the full-state Dijkstra solution with the Reduced-State Dijkstra algorithm. The results are shown in Figure 7. The full-state Dijkstra algorithm takes some two hours to complete, as opposed to 126 seconds for the Reduced-State Dijkstra. The cost degradation of the latter is less than 3%. This gives close to 60-fold speedup, with a minor loss in performance. Recall that the Reduced-State algorithms reduces the state space cardinality by a nominal factor of $m = 40$, the number of neighbors of each point as computed above. This is directly reflected in the reduction in computation time. The two algorithms were further compared for various problem parameters, with similar conclusions. In view of its efficiency, the Reduced-State Dijkstra is used as the underlying algorithm in the remainder of our simulation experiments.

We next examine briefly the effect of varying the weights of the cost function components. Figure 9 depicts the results for different weights of the altitude cost as compared to the riding qualities cost. The first run is conducted with unit weights to both cost criteria, resulting in the reference trajectory. The second run is conducted with the altitude weight, w_{altitude}, increased four-fold. Finally, the third run is conducted with the riding qualities weight, $w_{\text{riding\_qualities}}$, increased four-fold.

At the top left we show a 3D view of the trajectories. The main figure is a horizontal projection of these trajectories. The bottom right shows a 'stretched' sidewise projection (altitude versus range-to-go) of the trajectories. We can see that the riding-qualities-aware trajectory offers a smooth ride and does not follow closely the finer features of the terrain contour. The reference trajectory descends into such 'valleys' to a certain degree, while the altitude-aware trajectory dives deep into them. It also important to notice, looking at the top view of the trajectories, that every trajectory passes through a slightly different area of the map, hence the terrain under each trajectory is not the same. Overall, we achieve a smoothing effect, where the weights of the different cost criteria enable us to tune the degree of trajectory smoothness. Regarding the difference in run-time, the altitude-aware trajectory barely probes into the vertical degree of freedom – thus concluding the search more rapidly.

Next, we consider the A^* variant of the algorithm with the time-to-go heuristic. Figure 9 shows the effect of different time heuristic factors. The heuristic is taken to be $h(X) = wT_g(X)$, where T_g is the straight-line underestimate of the remaining time to the goal position. The algorithm was tested with weights $w = 0.3$, 1, 2, 4, to which we refer as the slow, medium, fast and faster heuristics. All runs were conducted with the Reduced-State A^* algorithm. Figure 9 depicts the obtained trajectories, and Table 2 summarizes the run-times and cost degradation (compared to the Reduced-State Dijkstra reference) for these runs.

The two smaller weights (slow and medium) give rise to an underestimating heuristic, and should not affect the optimal solution. However, they may affect the results of the (suboptimal) Reduced-State Dijkstra. It is indeed seen that the slow heuristic did not modify the quality of the obtained solution, while the medium heuristic led to a slight degradation of the cost, while the computation time was about halved. The two more aggressive heuristics, on the other hand, led the trajectory to a different area of the map, with a considerable cost degradation. Thus, despite some added speedup, we consider them to be unacceptable.

Algorithm	**Cost degradation**	**Run-time** [sec]
Reduced-State Dijkstra	Reference	126 sec
Reduced-State A* (slow)	0.0%	71 sec
Reduced-State A* (medium)	0.8%	56 sec
Reduced-State A* (fast)	22.1%	47 sec
Reduced-State A* (faster)	48.5%	14 sec

Table 2: A* heuristics comparison

The *hierarchical* variant of the Dijkstra algorithm is considered next. The hierarchical algorithm has two important parameters: the degree of down-sampling of the original grid and the width of the corridor to explore around the top-level trajectory. The corridor concept is illustrated in Figure 10. In Figure 10 we use a down-sampling of $k = 2$ in each axis. The algorithm first performs a search in the coarser $(60 \times 60 \times 20)$ map. It then returns to the original map, opens a corridor 7 vertices wide on each side of the coarse level trajectory, and performs a detailed search in that corridor. The detailed search was not similarly constrained in the vertical axis. It is important to emphasize that the same basic algorithm is used for both hierarchical levels, namely The Reduced-State Dijkstra is used at both hierarchical levels.

The total run-time of the Hierarchical Reduced-State Dijkstra algorithm (down-sample: 2, corridor: 7) is 57 seconds, while the cost degradation is 9.1%. The top-level trajectory (diagonals)

is found in just a few seconds, and the remaining time is used to find the detailed trajectory. It may be seen that the corridor around the coarse-level solution is not wide enough to encompass the optimal solution, so that the latter is not found by the finer search.

We next consider the effect of the two parameters of the hierarchical approach. First, we hold the down-sampling of the map at a fixed value of 2, and consider several corridor widths. This is shown in Figure 11. As expected, increasing the width of the corridor improves the accuracy of the hierarchical algorithm, and vice-versa. In particular, a corridor of 10 vertices gives a close-to-optimal performance. Table 3 summarizes the run-times and cost degradation (compared to the Reduced-State Dijkstra reference) of different corridor parameters presented in Figure 11.

Algorithm	**Cost degradation**	**Run-time** [sec]
Reduced-State Dijkstra	Reference	126 sec
Hierarchical Reduced-State Dijkstra (20)	0.0%	83 sec
Hierarchical Reduced-State Dijkstra (10)	0.9%	63 sec
Hierarchical Reduced-State Dijkstra (7)	9.1%	57 sec
Hierarchical Reduced-State Dijkstra (3)	20.7%	40 sec

Table 3: Hierarchical Dijkstra – performance for different corridor parameters

Next, we consider different down-sampling values while holding the corridor parameter constant at 10 vertices to each side. The results are shown in Figure 12; the time improvement due to increased down-sampling is marginal, as most of the times is spent on the detailed find-level search. However, down-sampling by 4 or more moves the search to a different area of the map altogether, and the solution deviates to an unacceptable degree from the optimal. Table 4 summarizes the run-times and cost degradation (compared to the Hierarchical Reduced-State Dijkstra with down-sampling of 2) of the different down-sampling parameters presented in Figure 12.

Algorithm	**Cost degradation**	**Run-time** [sec]
Hierarchical Reduced-State Dijkstra (2)	Reference	63 sec
Hierarchical Reduced-State Dijkstra (3)	1.4%	55 sec
Hierarchical Reduced-State Dijkstra (4)	21.9%	67 sec

Table 4: Hierarchical Dijkstra - performance for different down-sampling factors

The main results obtained in the above-described experiments are summarized in Table 5, to give a more complete perspective of the tradeoff of run-time speedup versus cost degradation. According to our previous findings, the A^* scheme utilizes the *medium* time heuristic, and the hierarchical algorithm uses a corridor parameter of 10 vertices to each side and a down-sampling

factor of 3. The last entry refers to a combined use of the A^* and hierarchical variants, yielding the fastest run time of 29 seconds, with mild performance degradation.

Algorithm	**Cost degradation**	**Run-time** [sec]
Full-state Dijkstra	Reference	7595 sec
Reduced-State Dijkstra	2.9%	126 sec
Reduced-State A*	3.7%	56 sec
Hierarchical Reduced-State Dijkstra	4.3%	55 sec
Hierarchical Reduced-State A*	4.4%	29 sec

Table 5: Run-time vs. cost degradation comparison of main algorithms

Obviously, the use of a more powerful computer and judicious programming will further improve the run-time. These results clearly indicate the potential for real-time, on-board, applicability.

6 Conclusion

This paper examines the applicability of global graph-search techniques to realistic trajectory optimization problems for General Aviation applications. A central element of our model is the inclusion of acceleration-dependent cost, which accounts for passenger comfort (or riding qualities) and pilot workload. The complexity of the suggested algorithms was examined with respect to the goal of obtaining computation times that allow the use of the algorithm for on-board trajectory planning, namely in the sub-minute area. Our results indicate that this goal is indeed feasible, by combining the computational acceleration methods that were considered.

Global search techniques enjoy some distinct advantages. The basic Dijkstra algorithm indeed performs an exhaustive search over all possible paths, and consequently obtains the globally optimal solution (up to the discretization error introduced in the discrete model setup). Even the reduced-state and other sub-optimal acceleration schemes that were proposed perform an exhaustive search in Cartesian space, although with some restrictions on angular configurations. As a consequence these algorithms do not need initialization with a candidate path or multiple runs with different initial condition, but rather obtain their best estimate for the global solution after a single run. Unstructured hard constraints that may induce multiple local optima, such as forbidden flight zones, are easily accommodated and merely reduce the search space. Moreover, the suggested algorithms are finite in nature, namely they terminate with their solution after a finite number of steps. Thus, the computation time can be strictly bounded, and the question of convergence does not arise at

all.

The reduction to a graph-search problem is obtained by discretizing the state space at some finite resolution. The selected resolution will obviously have a major impact on the accuracy of the obtained solution, as well as on computation time. We have outlined here some basic considerations that may lead to an appropriate choice of resolution for the intended application, so that the quantitative qualities of the continuous model are retained.

Many issues remain for further research. It would obviously be useful to obtain a theoretical handle on the approximation errors of the proposed algorithms, both those introduced due to the finite grid discretization, and those that are due to the sub-optimality of the accelerated search algorithms. These should be accompanied by additional numerical experimentation. It would also be interesting to compare the results obtained here with standard solvers based on Nonlinear Programming, using direct or indirect methods. It should be appreciated, however, that these algorithms are essentially local in nature, so that they may not be suitable for problems with multiple local minima. The combination of global search methods with local refinement using other (local) algorithms poses another potentially interesting that requires further inquiry.

The graph search algorithms themselves may obviously be further enhanced. For example, the search may proceed simultaneously from both ends of the trajectory, and terminate when these two parts meet. The effect of this and other possible enhancements should be evaluated empirically.

The discretization scheme that was used in this paper considers only trajectories that pass through the discrete grid points. As we mentioned in the Introduction, some recent Dijkstra-like algorithms for the continuous shortest-path problems rely on more elaborate discretization schemes that allow paths that proceed in-between grid points [25, 26], and therefore have the potential for better accuracy for a given grid resolution. It would be therefore interesting to extend these methods to include the dynamic aspects of our problem, namely the acceleration-related costs and constraints. From the end-user viewpoint, further work is required concerning the proper choice of cost function. This includes also the quantification of the pilot workload cost that was not elaborated in the present paper.

Acknowledgements

We appreciate the valuable comments, provided by Professor J.L. Speyer and his research team and by Professor M. Feron and his seminar group – during informal presentations of this research at UCLA and MIT, respectively. We would like to thank the Associate Editor and the three referees for their helpful comments. This research was supported by the Fund for the Promotion of Research at the Technion.

References

[1] Hewitt C., Henley A. J. and Boyes J. D., "A Ground and Obstacle Collision Avoidance Technique (GOCAT)", *IEEE Aerospace and Electronic Systems Magazine,* Vol. 6, Number 8, 1991, pp. 13–20.

[2] Tymczyszyn B. and Wilson G., "Development and flight testing of a general/corporate aviation terrain awareness and warning system," 2002 Report to the Aerospace Profession; 46th Society of Experimental Test Pilots (SETP) Symposium Proceedings, Los Angeles, CA, 2002, pp. 94-107.

[3] Clarke K. C., *Getting Started with Geographic Information Systems*, Prentice-Hall, New Jersey, 1990 (pp. 34–137).

[4] Fountain J. R., "Digital Terrain Systems", *IEE Colloquim on Airborne Navigation Systems Workshop*, (Digest No. 1997/169), 1997, pp. 4/1-4/6.

[5] Bryson A. E., *Dynamic Optimization*, Addison Wesley, Menlo Park, CA, 1999.

[6] Bryson A. E. and Ho Y. C., *Applied Optimal Control*, Wiley, New York, 1975 (pp. 1–127).

[7] Bertsekas D. P., *Dynamic Programming and Optimal Control*, 2nd Ed., Athena Scientific, Belmont, MA, 2000.

[8] Betts J. T., "Survey of Numerical Methods for Trajectory optimization", *Journal of Guidance, Control and Dynamics*, Vol. 21, No. 2, March-April 1998, pp. 193–207.

[9] Kushner H. J. and Dupuis P. G., *Numerical Methods for Stochastic Control Problems in Continuous Time*, Springer, New York, 1992.

[10] Funk J. E., "Optimal-Path Precision Terrain-Following System" *Journal of Aircraft*, Vol. 14, No. 2, 1977, pp. 128–134.

[11] Hargraves C. R. and Paris S. W., "Direct Trajectory Optimization Using Nonlinear Programming and Collocation", *Journal of Guidance, Control and Dynamics*, Vol. 10, No. 4, 1998, pp. 338–342.

[12] Asseo S. J., "Terrain Following/Terrain Avoidance Path Optimization Using the Method of Steepest Descent", *Proceedings of the IEEE National Aerospace and Electronics Conference*, vol. 3, IEEE, New York, 1988, pp. 1128–1136.

[13] Menon P. K. A., Kim E. and Cheng V. H. L., "Optimal Trajectory Synthesis for Terrain-Following Flight", *Journal of Guidance, Control and Dynamics*, Vol. 14, No. 4, 1991, pp. 807–813.

[14] Lu P. and Pierson B. L., "Optimal Aircraft Terrain-Following Analysis and Trajectory Generation", *Journal of Guidance, Control and Dynamics*, Vol. 18, No. 3, 1995, pp. 555–560.

[15] Lu P., "Inverse Dynamics Approach to Trajectory Optimization for an Aerospace Plane", *Journal of Guidance, Control and Dynamics*, Vol. 16, No. 4, 1993, pp. 726–732.

[16] Spong M. W., Thorp J. S. and Kleinwaks J. M. M., "The Control of Robot Manipulators with Bounded Input", *IEEE Transactions on Automatic Control*, Vol. AC-31, No. 6, 1986, pp. 483–490.

[17] Lee S. M., Bien Z. and Park S. O., "On-Line Optimal Terrain-Tracking System", *Optimal Control Applications and Methods*, Vol. 11, No. 4, 1989, pp. 289–306.

[18] Rehbock V., Teo K. L. and Jennings L. S., "A Linear Programing Approach to On-Line Constrained Optimal Terrain-Tracking Systems", *Optimal Control Applications and Methods*, Vol. 14, No. 4, 1993, pp. 229–241.

[19] Barnard R., "Terrain Tracking Based On Optimal Aim Strategies", *Optimal Control Applications and Methods*, Vol. 15, No. 2, 1994, pp. 145–150.

[20] Hess R. A. and Jung Y. C., "Generalized Predictive Control of Dynamic Systems", *Proceedings of the 1988 IEEE International Conference on Systems, Man and Cybernetics*, 1988, pp. 844–849.

[21] Jung Y. C. and Hess R. A., "Precise Flight-Path Control Using a Predictive Algorithm", *Journal of Guidance, Control and Dynamics*, Vol. 14, No. 5, 1991, pp. 936–942.

[22] Lu P., "Nonlinear Predictive Controllers for Continuous Systems", *Journal of Guidance, Control and Dynamics*, Vol. 17, No. 3, 1994, pp. 553–560.

[23] Lu P. and Pierson B. L., "Aircraft Terrain-Following based on a Nonlinear Continuous Predictive Control Approach", *Journal of Guidance, Control and Dynamics*, Vol. 18, No. 4, 1995, pp. 817–823.

[24] Mitchell J. S. B. and Keirsey D. M., "Planning strategic paths through variable terrain data", SPIE Vol. 485: Applications of Artifical Intelligence, 1984, pp. 172–179.

[25] Tsitsiklis J. N., "Efficient Algorithms for Globally Optimal Trajectories", *IEEE Transactions on Automatic Control*, Vol. 40, No. 9, September 1995, pp. 1528-1538.

[26] Sethian J. A., *Level Set Methods and Fast Marching Methods,* 2nd ed., Cambridge University Press, 1999.

[27] Dijkstra E. W., "A Note on Two Problems in Connection with Graphs", *Numerische Mathematik*, 1959.

[28] Cormen T. H., Leiserson C. E., and Rivest R. L., *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001, pp. 527–531.

[29] Nilsson N. J., *Principles of Artificial Intelligence*, Tioga Publishing Co., 1980, pp. 72–88.

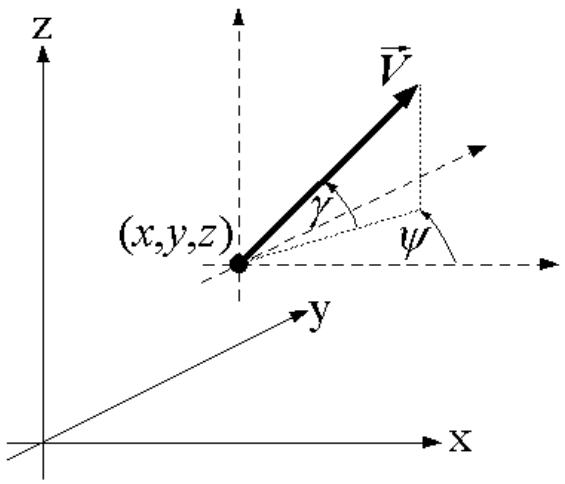

Figure 1: Airspace position and orientation

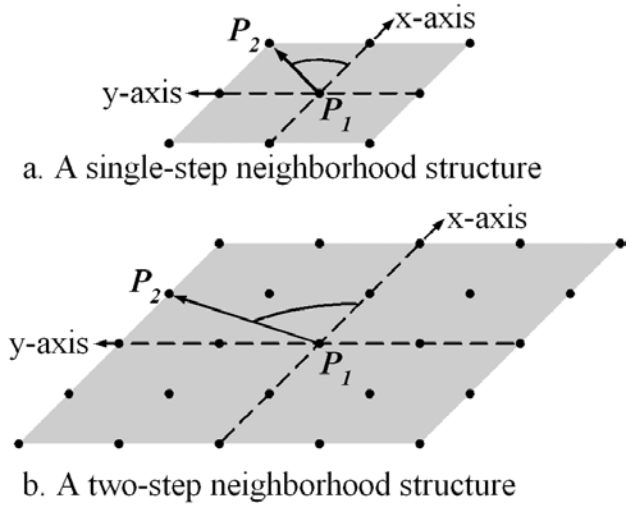

Figure 2: Horizontal-plane neighbors of a grid point

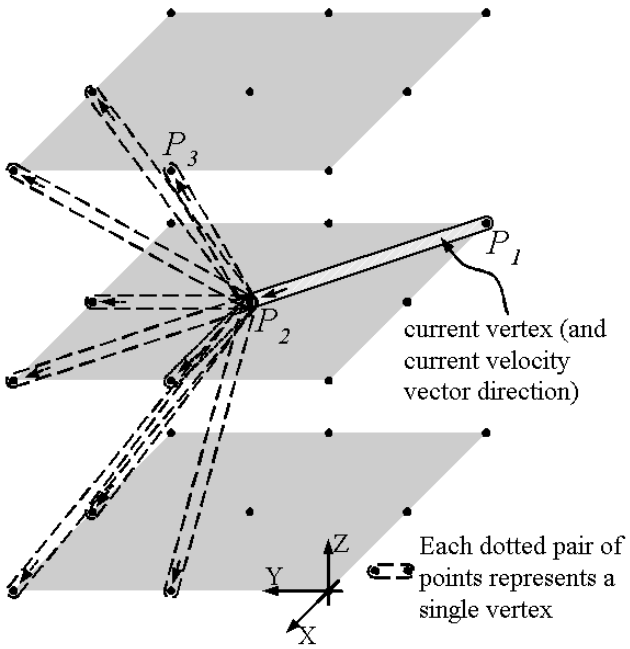

Figure 3: A vertex and its feasible neighbors

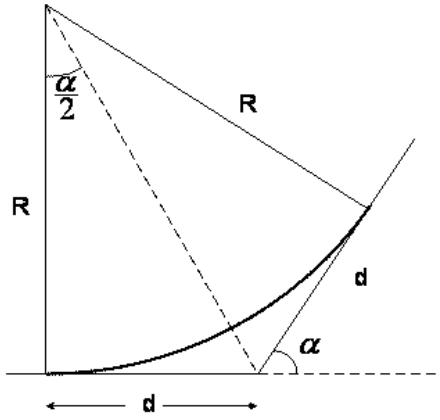

Figure 4: A trajectory segment

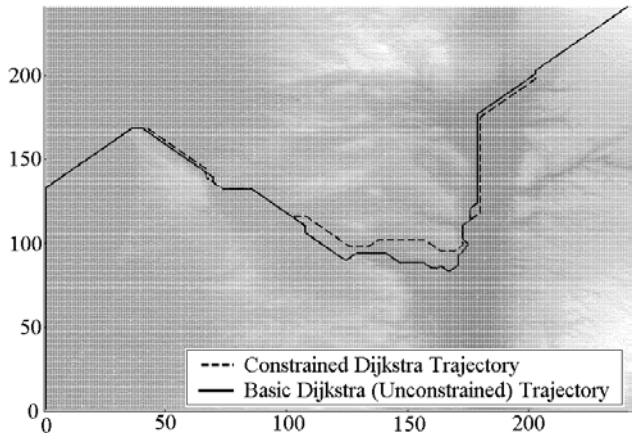

Figure 5: Basic vs. Constrained Dijkstra trajectories – top view. Darker parts of the map indicate lower ground levels.

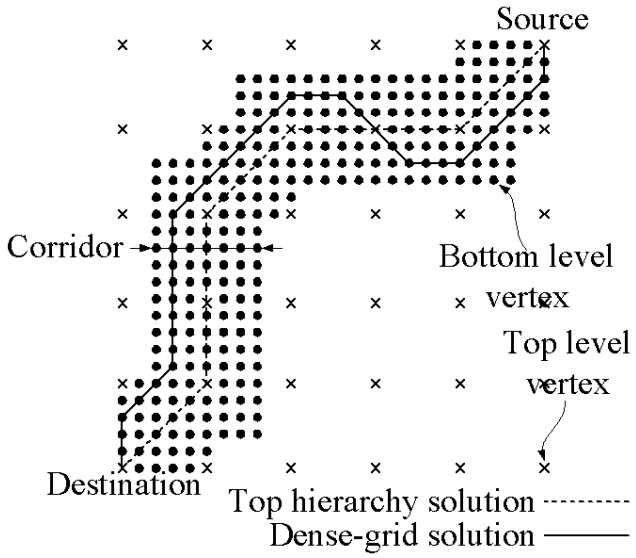

Figure 6: Illustration of the hierarchical scheme

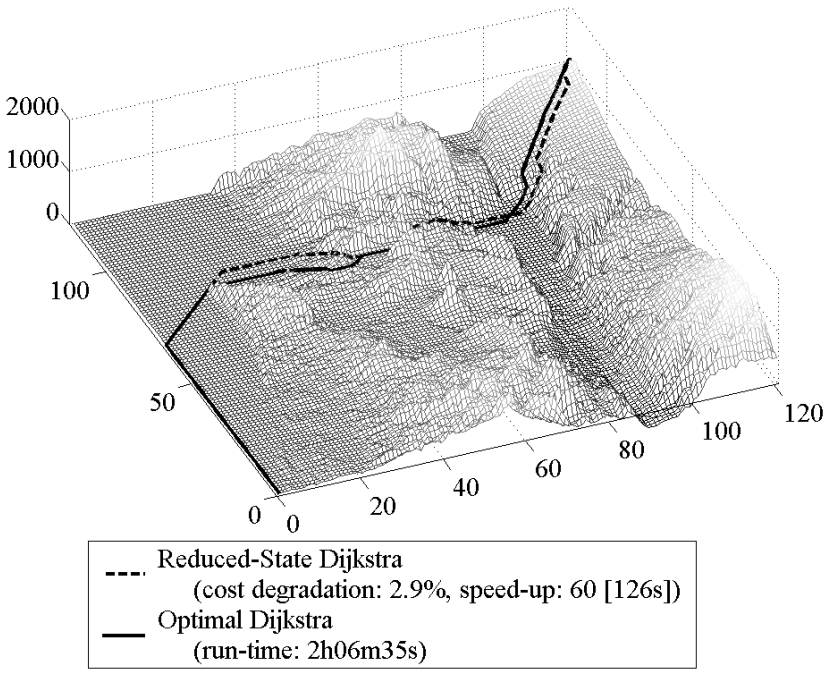

Figure 7: Full vs. Reduced-State Dijkstra

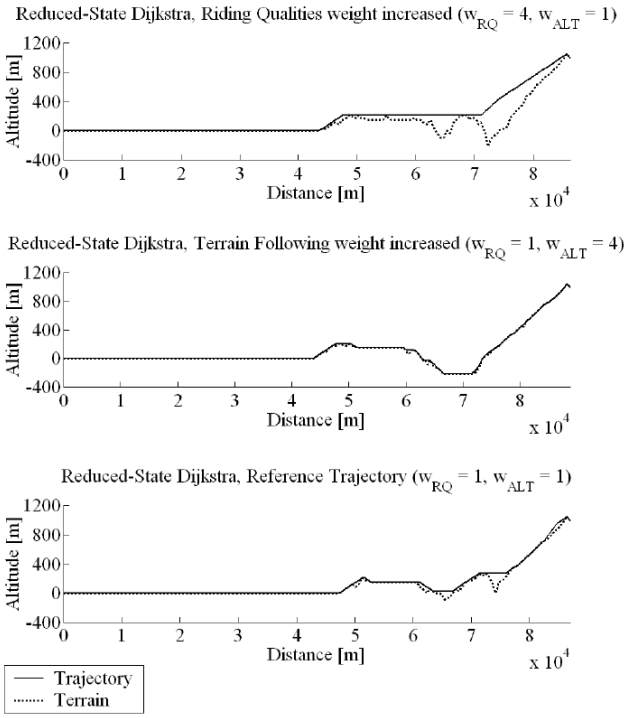

Figure 8: Cost Function Criteria – Trajectory Smoothness Tuning

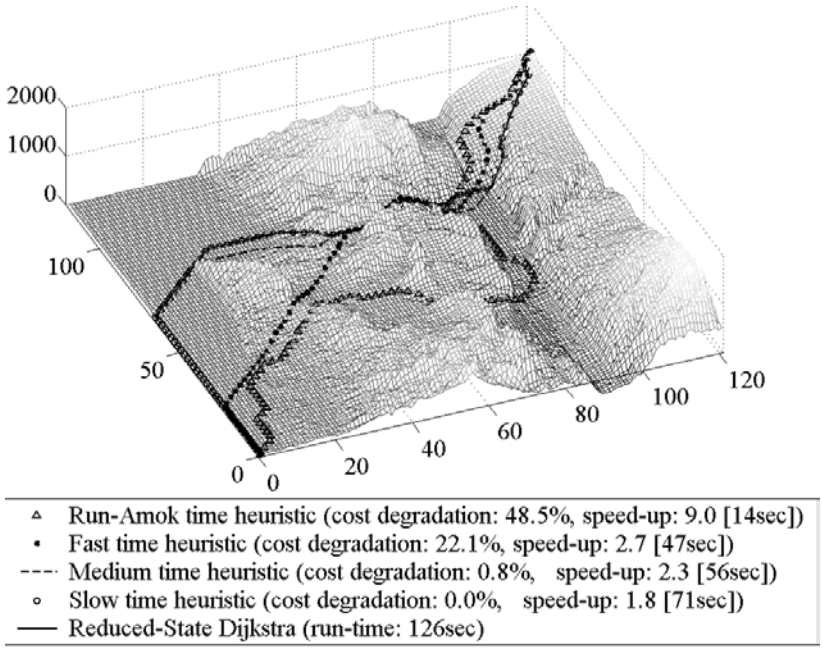

Figure 9: The A* time heuristic within the Reduced-State Dijkstra

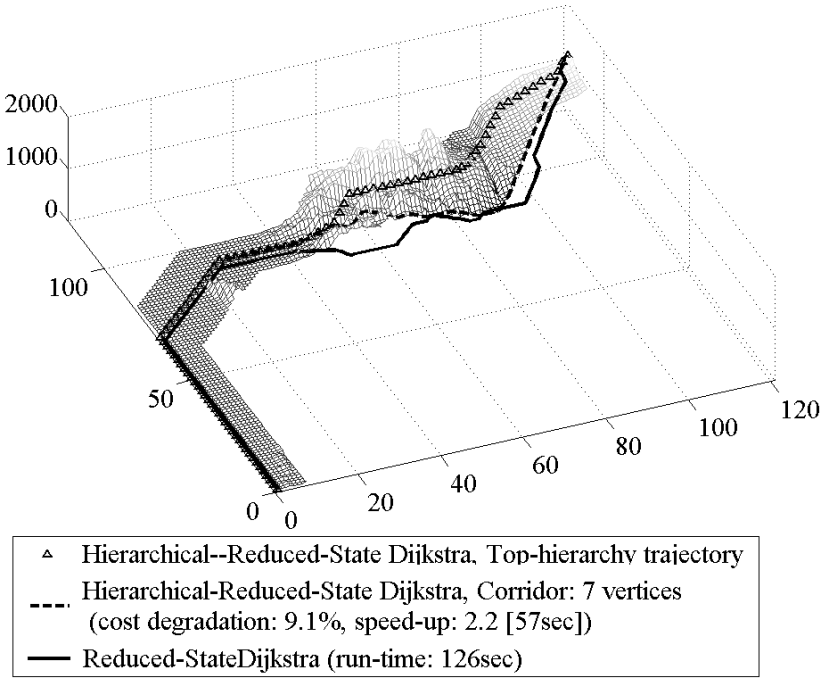

Figure 10: The hierarchical algorithm – illustration of the corridor concept

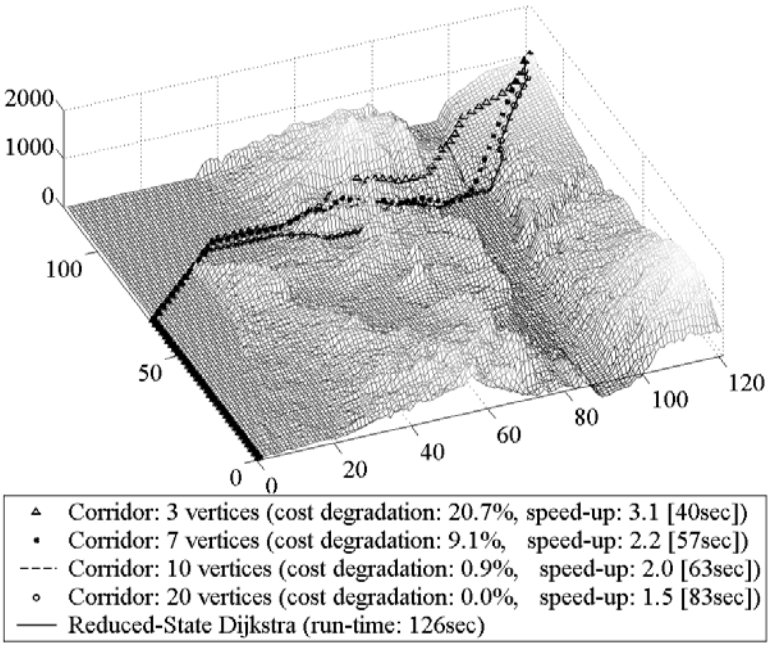

Figure 11: Hierarchical Reduced-State Dijkstra – different corridor widths

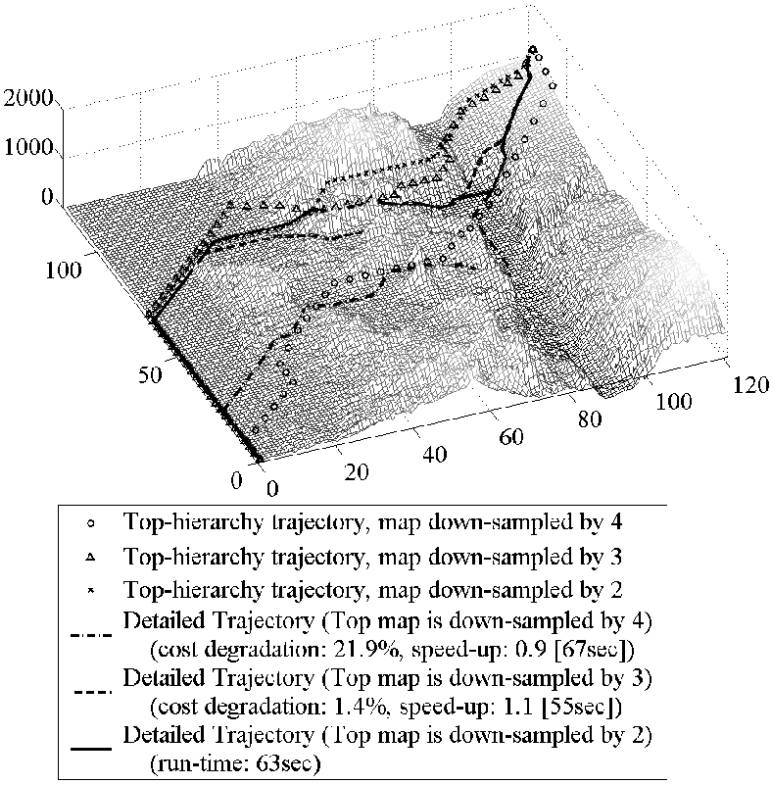

Figure 12: Hierarchical Reduced-State Dijkstra – different down-sampling factors