STOCHASTIC WEATHER ROUTING FOR SAILING VESSELS

Toby Allsopp
A thesis submitted in partial fulfilment of the requirements for the degree of Master of Engineering
Abstract

We consider the problem of finding the route that minimises the expected travel time between two points on the ocean under uncertain weather conditions. This has applications in long, offshore yacht racing. The uncertainty in the weather is modelled by a branching scenario tree in order to capture the serial correlation inherent in the evolution of weather systems over time. The case of deterministic weather conditions is considered using a dynamic programming algorithm. Isochrones are generated from the intermediate data produced by this algorithm by contouring a fitted bilinear time field function. The stochastic solution method is derived from the recursion used in the deterministic method, extended to include scenario as a state variable, yielding a stochastic dynamic programming algorithm. The solution obtained from the stochastic method is compared to that obtained from the deterministic method for a simple example of uncertain weather and is found to be superior. The computation time required to generate solutions using the stochastic method is investigated and found to be unacceptably long without modification. Two such modifications are found to reduce the solution time sufficiently without significantly compromising the quality of the solution.
Acknowledgments

The work described in this thesis has been funded by the Foundation for Research, Science and Technology, through the Graduate Research in Industry Fellowship programme. I would like to thank FRST for giving me the opportunity to not only do such interesting work, but to get paid for it.

Thanks must also go to my supervisors at the University of Auckland – Andy Philpott and Andrew Mason. Their advice and support made this project both successful and enjoyable.

The involvement of KiwiTech Marine Solutions Ltd is crucial to not only the project’s inception, but to its future. I am grateful for the opportunity to work with such switched on people, and look forward to continuing to do so.

I would also like to thank my mother and her partner for providing me with a place to live (and at such a reasonable rate) during the work towards this Masters degree.
# Contents

1. **Introduction** ........................................................................................................... 1  
   1.1 Previous Work ........................................................................................................ 1  
   1.2 Overview of Thesis ................................................................................................. 2  

2. **Physical Model** ...................................................................................................... 4  
   2.1 Co-ordinate System ............................................................................................... 4  
   2.1.1 Distance ............................................................................................................ 6  
   2.2 Problem Definition ............................................................................................... 8  
   2.2.1 Speed ................................................................................................................ 10  
   2.2.2 Wind .................................................................................................................. 11  
   2.2.3 Velocity Prediction ............................................................................................ 12  
   2.2.4 Current ............................................................................................................. 14  
   2.3 Weather Information ............................................................................................ 21  
   2.3.1 Wind .................................................................................................................. 21  
   2.3.2 Current ............................................................................................................. 24  
   2.4 Summary .............................................................................................................. 24  

3. **Deterministic Routing** ............................................................................................ 25  
   3.1 Network Definition ............................................................................................... 25  
   3.2 Dynamic Programming ......................................................................................... 28  
   3.2.1 Recursion .......................................................................................................... 34  
   3.2.2 Backward-looking ............................................................................................. 35  
   3.3 Graph generation ................................................................................................. 39  
   3.3.1 Graph Domain ................................................................................................. 39  
   3.3.2 Graph Resolution ............................................................................................ 41  
   3.4 Summary .............................................................................................................. 47  

4. **Isochrones** .............................................................................................................. 49  
   4.1 Isochrone Generation ......................................................................................... 49  
   4.2 Summary .............................................................................................................. 56  

5. **Stochastic Routing** ............................................................................................... 57  
   5.1 Stochastic Model ................................................................................................. 57  
   5.1.1 Climatology ...................................................................................................... 62  
   5.1.2 Problem Definition ......................................................................................... 62  
   5.2 Solution Method .................................................................................................. 64  
   5.2.1 Implementation ............................................................................................... 66  
   5.3 Summary .............................................................................................................. 78  

6. **Comparison of Deterministic and Stochastic Solutions** ................................... 79  
   6.1 Summary .............................................................................................................. 83  

7. **Computation Time Improvements** ....................................................................... 84
7.1 Scanning Order ................................................................. 88
7.2 Scanning Selection ........................................................... 90
  7.2.1 Constant Angle Resolution ............................................. 90
  7.2.2 Distance ................................................................. 94
  7.2.3 Global Optimisation .................................................... 97
7.3 Summary ................................................................. 105

8. Conclusions ........................................................................ 106
  8.1 Future Work ................................................................. 107

9. References .......................................................................... 109
1. Introduction

Weather routing is the process of determining an optimal route to sail between two given points, starting at a given time, based on predictions of what the weather will be. The focus of this thesis is long, offshore yacht races, such as the Whitbread Round The World Race (the Whitbread), in which an optimal route is one that takes the minimum time. These races, most notably the Whitbread, can be high profile, so sponsors are prepared to spend large amounts of money to give their yacht more chance of winning and thus obtain more publicity. There is therefore some motivation for companies to produce and sell software that purports to increase this chance.

The goal of the work described in this thesis is the production of software for generating optimal (minimal time) routes for use in races such as the Whitbread. The work has been carried out in conjunction with KiwiTech Marine Solutions Limited (KiwiTech), a New Zealand company specialising in marine software and hardware. KiwiTech currently sells a software package that includes a weather routing module. This software, RaceTech, was used by six of the ten yachts in the 1997-98 Whitbread. The weather routing aspect of this software performs what we will refer to as deterministic weather routing, that is it only considers one possibility for the weather and produces an optimal route based on the assumption that the predicted weather will happen exactly.

However, races such as the Whitbread take place over large areas of ocean and can take months to complete. It is a very difficult problem to predict the weather for this sort of length of time with any accuracy, so an approach based on a single forecast can yield solutions that perform badly when implemented under real weather conditions. The idea behind this work is that we should allow the possibility of different weather conditions evolving in the future, and produce routes that perform well under all of them, on average. We will refer to such an approach that takes into account uncertain weather conditions as stochastic weather routing.

1.1 Previous Work

There exists a large body of work dealing with the problem of optimal ship routing, in which large, powered ships are considered. In these problems, sea-state, that is the direction and magnitude of waves, is the determining factor in the speed at which a ship
can travel in a given direction. This differs from the sailing vessel problem in which a yacht’s speed in a given direction is determined by the speed and direction of the wind. Also, different objective functions are often used, such as fuel consumption and damage to cargo or the ship.

The approaches in the literature can be divided into two categories – those based on a network discretisation of the problem domain and those based on applications of the calculus of variations or optimal control theory. Hanssen and James (1960) proposed a scheme for the case when weather conditions do not vary with time using the sequential construction of lines of equal time, or isochrones. The problem considered was necessarily that of minimal time. The same problem was addressed by Haltiner, Hamilton, and Árnason (1962) using the calculus of variations. Faulkner (1963) extended this approach to the case of known, time-varying sea-state conditions. Numerous variations on this type of method have been presented, for example Bijlsma (1975), who also considered minimal fuel consumption, and Papadakis and Perakis (1990).

The other approach was taken by Zoppoli (1972), who treated the minimal time problem as a decision process solvable using dynamic programming by discretising the problem domain. Zoppoli also allowed for the possibility of uncertain weather by treating the sea-state conditions as random variables. A dynamic programming approach was also used by Carryer (1995), applied to the problem of yacht racing optimisation. This work was based on the work of Leyland (1992), who dealt with solar powered car racing. Both Leyland and Carryer used the progressive hedging algorithm for scenario aggregation to extend the deterministic solution method to the stochastic case.

This work uses a similar dynamic programming method for the deterministic case, but uses a stochastic dynamic programming approach, similar in concept to Zoppoli (1972), for the stochastic case. However, rather than treating the weather as a random variable, we use a branching scenario structure in order to model the manner in which weather evolves over time.

1.2 Overview of Thesis

In Chapter 2 of the thesis, we describe the physical models used to describe the problem. This includes a description of the coordinate system used to describe positions
on the surface of the earth, the formulae used to determine the speed of a yacht in given weather conditions, and the form in which weather information is stored and the way in which it is used.

We then, in Chapter 3, proceed to describe a dynamic programming algorithm for the solution of the deterministic problem. This introduces important concepts that are used in the adaptation of this algorithm for the stochastic case, in Chapter 5. A digression is made in Chapter 4 to describe a method for generating isochrones from the intermediate data produced by the deterministic solution method.

Chapter 5 deals with the case of uncertain weather. The model for serially correlated stochastic weather scenarios is described and the deterministic solution method is extended to produce the minimal expected time solution. The solution generated by this method is compared to the deterministic and minimal distance solutions for a simple example of uncertain weather in Chapter 6. The computing time needed to calculate the optimal solution using this algorithm is discussed in Chapter 7, and a number of methods for reducing this are described and tested.
2. Physical Models

This chapter describes the coordinate system we will use to identify points on the surface of the earth, and formulae for calculating distances and angles in this coordinate system. We also define the manner in which the performance of a yacht is characterised and derive formulae for determining a yacht’s speed in a given direction as a function of wind and current. The representation used for wind and current is presented.

2.1 Co-ordinate System

For the purposes of this work, we will approximate the earth by a sphere. In reality, the earth bulges at the equator and is flattened at the poles, due to its rotation, but considering this makes it extremely difficult to render any analytic description of quantities such as distance and angle on the surface. A concept used extensively in the analysis of problems on the surface of this sphere is that of the great circle. A great circle is the intersection of a plane passing through the centre of the sphere with the surface of the sphere, as illustrated in Figure 2.1.
Great circle

Plane

Centre of sphere

Sphere

Great circle

**Figure 2.1** A great circle is the intersection of a plane passing through the centre of a sphere with the surface of the sphere.

An important property of great circles is that a great circle that passes through two points on the surface of a sphere is the shortest path between those points on the surface of the sphere.

The position of a node on the earth’s surface is determined by its latitude and longitude (see Figure 2.2). The equator is a great circle that lies between the poles. This is the line of 0° latitude. Other lines, or parallels, of latitude are the intersections of planes parallel to the equator with the surface of the earth. The north pole is at 90°N and the south pole is at 90°S. For the purposes of computation, latitudes north of the equator are considered to be positive and those south of the equator are considered to be negative.

Lines, or meridians, of longitude are great semicircles perpendicular to the equator that terminate at each pole. The meridian of 0° longitude passes through Greenwich, England and is called the prime meridian. Other meridians of longitude are rotated about an axis parallel to the equator passing through the centre of the earth. The angle
through which the prime meridian is rotated, either west or east, specifies meridians of longitude. Longitude is measured up to 180° west or east. For the purposes of computation, west is considered to be positive and east is considered to be negative.

**Figure 2.2** Latitude and longitude coordinate system for specifying a location on the earth’s surface

Thus, a location on the earth’s surface is specified by a vector

\[
x = \begin{pmatrix} x_{\text{lon}} \\ x_{\text{lat}} \end{pmatrix}, -90^\circ \leq x_{\text{lon}} \leq 90^\circ, -180^\circ \leq x_{\text{lat}} \leq 180^\circ.
\]

It is useful to represent the surface of the earth in two dimensions, for example to display a map on paper or a computer screen. One of the most useful methods for doing this is the Mercator projection, in which parallels of latitude and meridians of longitude are perpendicular. In this projection, the most common for navigational maps, a course of constant heading appears as a straight line.

### 2.1.1 Distance

Because the earth is not flat, the shortest distance between two points on the earth’s surface is not a simple Euclidean distance. For the purposes of this work, the earth is assumed to be a sphere with circumference 21600 nautical miles (NM) or 40003.2km. Note that a nautical mile is the length of one minute (\(\frac{1}{60}\)) of arc on the surface of this sphere, and is defined to be 1852m. The angle, γ, subtended by two points, \(x_1\) and \(x_2\), on this sphere is given by either the Law of Cosines,
2.1 Co-ordinate System

\[ \gamma(x_1, x_2) = \cos^{-1}\left( \sin x_{1, lat} \sin x_{2, lat} + \cos x_{1, lat} \cos x_{2, lat} \cos(x_{1, lon} - x_{2, lon}) \right), \]  

(2.1a)

or the Haversine Formula,

\[ \gamma(x_1, x_2) = 2\sin^{-1} \sqrt{\sin^2 \frac{x_{1, lat} - x_{2, lat}}{2} + \cos x_{1, lat} \cos x_{2, lat} \sin^2 \frac{x_{1, lon} - x_{2, lon}}{2}}. \]  

(2.1b)

These formulae are taken from *The Geographic Information Systems FAQ* (Nyman, 1998). The second of these equations (2.1b) is less susceptible to rounding error for small distances than the first (2.1a). Figure 2.3 shows the relative error in the distance computed using (2.1a), where the value computed using (2.1b) is assumed to be correct.

![Rounding error introduced for small distances](image)

**Figure 2.3** Error in distance computation when using the Law of Cosines formula (2.1a) rather than the Haversine formula (2.1b).

It can be seen that for distances less than approximately 50cm there is significant error introduced by using (2.1a). However, distances of this magnitude are of little interest for weather routing. Indeed, civilian Global Positioning System receivers, as commonly used for marine navigation, cannot measure position to within less than approximately 100m (Bennett and Nakamura, 1997). Thus, the accuracy of distances less than 100m is moot, and (2.1a) is used because it is less computationally expensive.
If we work in radians then the length of the arc that subtends the angle \( \gamma \) is simply the radius of the sphere multiplied by the angle, that is

\[
d_{GC}(x_1, x_2) = R_{\text{earth}} \gamma(x_1, x_2),
\]

where \( R_{\text{earth}} \) is the radius of the earth and \( d_{GC}(x_1, x_2) \) is the great circle distance from \( x_1 \) to \( x_2 \), in the same units as \( R_{\text{earth}} \). Note that \( R_{\text{earth}} = \frac{40003.2}{2\pi} \text{ km} = \frac{21600}{2\pi} \text{ NM} \).

This is the distance if \( x_2 \) is reached from \( x_1 \) by following a great circle. The direction in which to head in order to follow this great circle is given, in Nyman (1998), by

\[
\theta_{GC}(x_1, x_2) = \begin{cases} 
\theta, & \sin(x_2^{\text{lon}} - x_1^{\text{lon}}) < 0 \\
2\pi - \theta, & \text{otherwise}
\end{cases},
\]

where

\[
\theta = \cos^{-1} \left( \frac{\sin x_2^{\text{lat}} - \sin x_1^{\text{lat}} \cos \gamma(x_1, x_2)}{\sin \gamma(x_1, x_2) \cos x_1^{\text{lat}}} \right).
\]

This is the initial true course, where true course is the angle, measured clockwise, made with the local meridian. In general, the true course is not constant along a great circle.

### 2.2 Problem Definition

We wish to minimise the time taken to sail between two points, \( x_{\text{start}} \) and \( x_{\text{finish}} \), starting at time \( t_{\text{start}} \). This can be regarded as an optimal control problem, as has been done by a number of authors, for example Faulkner (1963). Although we do not consider solving the problem in this manner, it is useful to at least formulate it. Given the initial condition, \( x(t_{\text{start}}) = x_{\text{start}} \), and the velocity function \( v(x, \theta, t) \), a path is completely determined by the control function \( \theta(t), t \in (t_{\text{start}}, T) \), where \( \theta(t) \) is the direction in which the boat should move at time \( t \) and \( T \) is the time at which the boat reaches \( x_{\text{finish}} \). The relationship between the path and the control function can be expressed recursively as

\[
x(t + dt) = x(t) + v(x(t), \theta(t), t) dt, \quad t \in [t_{\text{start}}, T),
\]
or, non-recursively, as

\[ x(t) = x(t_{\text{start}}) + \int_{t_{\text{start}}}^{t} v(x(u), \theta(u), u) \, du, \quad t \in [t_{\text{start}}, T]. \] (2.2)

The problem now is to find \( \theta(t) \) to minimise \( T \) such that \( x(T) = x_{\text{finish}} \), where \( x(T) \) is defined by the choice of \( \theta(t) \) in (2.2). This problem is a special case of a problem in the calculus of variations known as the problem of Bolza. Methods for the solution of this problem have been suggested by, for example, Bijlsma (1975). However, the approach we consider does not solve for the control function \( \theta(t) \), but rather for \( x(t) \) directly.

Hence, we are interested in calculating the time taken for a given path. From this point of view, it makes little sense to regard a path as a function of time, because we do not know in advance the time at which the path will reach the destination or, indeed, any intermediate point (with the exception of the start point). Clearly, for a known start time, finding the time at which the destination is reached is exactly equivalent to finding the travel time for that path.

We will consider a path to be defined by position as a function of a parameter that ranges between zero and one, that is

\[ x(\tau), \quad 0 \leq \tau \leq 1, \quad x(0) = x_{\text{start}}, \quad x(1) = x_{\text{finish}}. \]

The direction in which the boat must move in order to achieve this path can be expressed as a function of the same parameter as

\[ \theta(\tau) = \lim_{\Delta \tau \to 0} \theta_{\text{GC}}(x(\tau), x(\tau + \Delta \tau)), \quad 0 \leq \tau \leq 1. \]

The time taken to sail from the start position \( x(0) = x_{\text{start}} \) to any other point on the path can be represented by the recursive integral

\[ t(\tau) = \begin{cases} \int_{0}^{\tau} \frac{d_{\text{GC}}(x(v), x(v + dv))}{\|v(\theta(v), x(v), t(v))\|}, & 0 < \tau \leq 1, \\ t_{\text{start}}, & \tau = 0. \end{cases} \]

A consequence of the \( t(v) \) term in the denominator of the integrand in this expression is that the time at which every previous point on the path is reached must be evaluated.
before that of the point of interest. This makes for a very expensive calculation in order to obtain the travel time for the path. We will approximate a path by a sequence of $n$ great circles joining $n+1$ points

$$x_i, \quad i = 1, \ldots, n+1, \quad x_1 = x_{\text{start}}, \quad x_{n+1} = x_{\text{finish}}.$$

This approximation alone does not offer significant benefits because the problem of finding the travel time of each of these great circle segments is exactly that of finding the travel time of the entire path. In order to simplify this calculation we will assume that the boat’s velocity is constant over each great circle segment. We can now approximate the travel time for a great circle segment by

$$c_{\text{seg}}(x_1, x_2, t) = \frac{d_{GC}(x_1, x_2)}{s(\theta_{GC}(x_1, x_2), x_1, t)}, \quad (2.3)$$

where $x_1$ and $x_2$ are the start and end points of the segment and $t$ is the time at which we start. The travel time of an entire path made up of $n$ of these great-circle segments, starting at time $t_{\text{start}}$, is given by $c_t(n, t_{\text{start}})$ where

$$c_t(j, t) = \begin{cases} 0, & j = 1 \\ c_t(j-1, t) + c_{\text{seg}}(x_{j-1}, x_j, t + c_t(j-1, t)), & 2 \leq j \leq n. \end{cases}$$

A consequence of this approximation is that it is desirable for all distances $d_{GC}(x_j, x_{j+1})$ to be as short as possible in order to reduce errors caused by boat speed varying between the points in the path.

### 2.2.1 Speed

The speed of a yacht is determined by many factors, including wind strength, true wind angle, current, waves, and sail settings. These can be broken down into environmental factors, such as wind and current, and controllable factors, such as sail choice and trim.

It is assumed that, for any given environmental conditions, the controllable factors will be set so as to maximise the speed of the yacht.
The only environmental factors considered in this model are wind and current. The reason other factors have been ignored is that information about how they affect the speed of a yacht is not readily available at present.

Where information about wind and current comes from and what form it takes will be discussed later in this section.

2.2.2 Wind
The single most important factor in determining the speed of a yacht is the angle at which it is sailing to the wind and the speed of the wind. The wind, \( \mathbf{w} \), is defined by a direction \( \text{arg} \mathbf{w} \) and a speed \( \|\mathbf{w}\| \) or, equivalently, by its vector components in the latitude and longitude directions, that is

\[
\mathbf{w} \equiv \left( \begin{pmatrix} w_{\text{lon}} \\ w_{\text{lat}} \end{pmatrix} \right) \equiv \|\mathbf{w}\| \angle \text{arg} \mathbf{w}.
\]

Wind varies with time \( t \) and position \( x \), that is \( \mathbf{w} = \mathbf{w}(x,t) \).

The actual direction in which the wind is going is not what determines the speed of the yacht, but rather the angle the heading of the yacht makes with the direction from which the wind is coming. The direction in which the yacht is pointed is the heading, \( \phi \), the direction from which the wind is coming is the true wind direction, \( \text{arg} \mathbf{w} \), and the angle between these two is the true wind angle, \( \alpha \). These terms are illustrated in Figure 2.4. These angles are qualified with the word ‘true’ to distinguish between the wind relative to land, the true wind, and the wind relative to the boat, the apparent wind.

![Figure 2.4 Heading, true wind direction and true wind angle](image)
2.2 Problem Definition

True wind angle, $\alpha$, is given by

$$\alpha(\text{arg } \mathbf{w}, \phi) = 180^\circ - \text{arg } \mathbf{w} - \phi.$$  \hspace{1cm} (2.4)

2.2.3 Velocity Prediction

In order to predict the velocity of a yacht for given environmental and controllable conditions, we must consider the forces acting on it. The calculation of these forces and their interactions is beyond the scope of this work, but a simplified overview is given as background.

Figure 2.5 shows the most important forces acting on a yacht, from the point of view of determining its velocity. The primary of these is that generated by the action of the wind on the sail. In the case of a yacht sailing upwind, that is with an acute true wind angle, the sail is used as an aerofoil and generates lift in a direction approximately perpendicular to the wind. This lift force can be regarded as having components in the direction in which the yacht is heading and in a direction perpendicular to this. The former is opposed to some degree by the drag force due to the resistance of the hull through the water; the remainder gives the yacht velocity in the direction in which it heading. The drag force due to the keel mostly opposes the latter. The sideways force not opposed by the keel force causes the velocity of the boat to be in a slightly different direction to it’s heading; this difference is called leeway. In this work it is assumed that effects of leeway are to be compensated for if the generated paths are to be used in practice, so these effects are not incorporated into the model.
The maximum speed at which a yacht can sail at a given true wind angle and true wind speed is difficult, if not impossible, to determine analytically. This information is instead generated by numerical velocity prediction programs and takes the form of a discrete set of true wind angle, true wind speed, and boat speed triples.

Interpolating between known values from the table can then approximate the predicted maximum speed for any true wind angle and true wind speed. A bilinear interpolation in true wind speed and true wind angle was used for this purpose.

This means that we have a description of the yacht’s speed $p_0$ as a function of true wind speed $\|w\|$ and true wind angle $\alpha$, that is

$$p_0 = p_0(\alpha, \|w\|).$$

Figure 2.6 shows such a function. This plot was produced by KiwiTech’s performance analysis software.
2.2 Problem Definition

It is convenient to consider the polar velocity as a function of heading $\phi$ so we will define

$$p(\phi, w) = p_0(\alpha(\arg w, \phi), \|w\|),$$

(2.5)

where $\alpha$ is defined as in (2.4). Combining this with the description of wind as a function of position and time, we get an expression for boat speed as a function of heading, position, and time, that is

$$p(\phi, w(x,t)) = p(\alpha(\arg w(x,t), \phi), \|w(x,t)\|).$$

Note that the speed predicted by the polar is assumed to be in the direction in which the yacht is heading, that is the effects of leeway are ignored, so the velocity predicted by the polar is

$$p(\phi, w(x,t)) = p(\phi, w(x,t)) \angle \phi.$$

2.2.4 Current

Current is the movement of the water relative to land. Current is a vector quantity similar to wind and we can write

Figure 2.6 Velocity prediction polar plot for true wind angles from 0° to 180° and true wind speeds from 0 to 20 knots.
\[ c \equiv \begin{pmatrix} c_{lon} \\ c_{lat} \end{pmatrix} = \|c\| \arg c. \]

Like wind, current varies with position and time, that is \( c = c(x, t) \).

Remembering that wind velocity is relative to land, it follows that a yacht in some current experiences a modified wind velocity. We will call this current-modified wind \( w^c \) and write

\[ w^c(x, t) = w(x, t) - c(x, t). \]

This is the velocity of the wind relative to the water. Applying (2.5) with this modified wind gives the yacht’s speed, \( p(\phi, w^c) \), in the direction \( \phi \) relative to the water. To get the velocity of the yacht relative to the land, \( v \), we must add the velocity of the water relative to the land, that is

\[ v(\phi, w, c) = p(\phi, w^c) + c. \]

An important consequence of the introduction of current is that the velocity of the yacht is no longer in the direction in which it is heading, \( \phi \). This is illustrated Figure 2.7.

\[ \text{Figure 2.7 Effect of current on wind and boat velocity} \]

When calculating the time taken to sail between two points, as in (2.3), the direction in which the boat should move, not the direction in which to head the boat, is specified by the two points. However, we have an expression for the velocity of the boat for a given heading. Because the speed is obtained from a table of numbers, it is not possible to
analytically invert this formula, so a numerical method must be employed. What we require is the set $\Phi$ of boat headings $\phi$ that give boat movement $v(\phi, w, c)$ in the desired direction $\theta$, that is

$$\Phi(\theta)_{w,c} = \{ \phi : \arg v(\phi, w, c) = \theta, -180^\circ \leq \phi < 180^\circ \}.$$  

This is the set of directions in which it is possible to head the boat and result in a velocity in the direction $\theta$. The reason there is a set of these directions is that there is potentially more than one direction in which to head that results in a velocity in the desired direction. An example of this case is shown in Figure 2.8.

**Figure 2.8** Different headings, $\phi_1$ and $\phi_2$, can result in the same velocity direction, $\theta$.

In this example, it is clear that $v(\phi_1, w, c)$ is a more desirable velocity than $v(\phi_2, w, c)$. When choosing the direction in which to head, we wish to select

$$\phi^*_{w,c} = \arg \max_{\phi \in \Phi(\theta)} \| v(\phi, w, c) \|,$$

where

$$\arg \max_{\phi \in \Phi(\theta)} \| v(\phi, w, c) \| = \phi^*_{w,c} \text{ such that } \| v(\phi^*, w, c) \| = \max_{\phi \in \Phi(\theta)} \| v(\phi, w, c) \|. $$
2.2 Problem Definition

Because the current vector, \( c \), is constant with respect to \( \phi \), it can be ignored when performing this maximisation and the projection of the polar (velocity prediction) component of the velocity, \( p(\phi, w^\phi) \), in the direction \( \theta \) can be used instead, giving

\[
\phi^*|_{w,c} = \arg \max_{\phi \in \Phi(\theta)} p(\phi, w^\phi) \cos(\theta - \phi).
\]  

(2.6)

It is now necessary to note that for angles \( \phi \in \Phi(\theta) \), \( p(\phi, w^\phi) \) satisfies

\[
p(\phi, w^\phi) \sin(\theta - \phi) = \|c\| \sin(\arg c - \theta) = h, \forall \phi \in \Phi(\theta),
\]  

(2.7)

where \( h \) is the projection of the current vector perpendicular to the direction in which we want to travel, \( \theta \). This is illustrated in Figure 2.9.

![Figure 2.9 Height of polar and current components, \( h \), must be equal.](image)

We can now rewrite (2.6) as
\[ \phi^* = \begin{cases} \arg \max_{\phi \in \Phi(\theta)} \frac{h \cos(\theta - \phi)}{\sin(\theta - \phi)}, & h \neq 0 \\ \arg \max_{\phi \in \Phi(\theta)} p(\phi, w^c) \cos(\theta - \phi), & h = 0 \end{cases} \]

\[ = \begin{cases} \arg \max_{\phi \in \Phi(\theta)} \frac{h}{\tan(\theta - \phi)}, & h \neq 0 \\ \arg \max_{\phi \in \Phi(\theta)} p(\phi, w^c) \cos(\theta - \phi), & h = 0 \end{cases}. \quad (2.8) \]

Note that, for the cases when \( h \neq 0 \), the selection of \( \phi^* \) is independent of the polar. This is because the value of \( h \) is constant for given current \( c \) and direction \( \theta \). It follows from (2.7) and the fact that \( p(\phi, w^c) \geq 0, \forall \phi \) that

\[ h > 0 \Rightarrow \sin(\theta - \phi) > 0 \Rightarrow 0^\circ < \theta - \phi < 180^\circ \]

\[ h < 0 \Rightarrow \sin(\theta - \phi) < 0 \Rightarrow -180^\circ < \theta - \phi < 0^\circ \]

\[ h = 0 \Rightarrow \sin(\theta - \phi) = 0 \Rightarrow \theta - \phi = 0^\circ \text{ or } \theta - \phi = 180^\circ \]

or \( p(\phi, w^c) = 0 \)

We will disregard the case of \( h = 0 \Rightarrow p(\phi, w^c) = 0 \) because in general there is no value of \( \phi \) for which this is true.
2.2 Problem Definition

The function \( \cot(\theta - \phi) \) is shown for the domain of interest in Figure 2.10. Combining (2.8) and (2.9) and observing the form of the cotangent function, it follows that, for \( h \neq 0 \),

\[
\phi^* = \arg \min_{\phi \in \Phi(\theta)} (\theta - \phi), \quad h > 0 \\
= \arg \max_{\phi \in \Phi(\theta)} (\theta - \phi), \quad h < 0 \\
= \arg \min_{\phi \in \Phi(\theta)} (\theta - \phi), \quad 0^\circ < \theta - \phi < 180^\circ \\
= \arg \max_{\phi \in \Phi(\theta)} (\theta - \phi), \quad -180^\circ < \theta - \phi < 0^\circ .
\]

(2.10)

When \( h = 0 \), it follows that \( (\theta - \phi) \in \{0^\circ, 180^\circ\} \) and we have
the desired direction, the heading closest to the desired direction gives the greatest velocity in that direction. Thus, it is only necessary to generate one element of the set \( \Phi(\theta) \) when searching for \( \phi^* \), provided that that element is the one closest to \( \theta \).

Using the definition of \( h \) in (2.7), it is possible to define the set of possible headings as

\[
\Phi(\theta)_{w,c} = \{ \phi : p(\phi) \cos(\theta - \phi) = h(\theta, c), \ -180^\circ \leq \phi < 180^\circ \}.
\]

Now \( \Phi(\theta) \), and thus \( \phi^* \), depends only on \( \theta \) and \( h \) rather than on \( \theta \), \( w \), and \( c \).

An expression for the speed of the yacht in a given direction \( \theta \) can now be formulated as

\[
s(\theta, x, t) = \| p(\phi^*(\theta, h(\theta, c(x, t))), w(x, t), c(x, t)) \|
\]

\[
= \| p(\phi^*(\theta, h(\theta, c(x, t))), w(x, t)) + c(x, t) \|^2,
\]

where \( h(\theta, c) = \| c \| \sin(\arg c - \theta) \).

It is possible that, for large currents, the set \( \Phi(\theta) \) is empty, meaning that it is not possible to sail in the direction \( \theta \). This can be interpreted as an infinite travel time for any segment for which this is the case.
2.3 Weather Information

2.3.1 Wind

The most widespread format for transmitting electronic weather information is known as a GRIB (GRIdded Binary) file. These files can contain information about many different aspects of the weather, such as true wind speed and direction, air pressure, air temperature and surface temperature. For more information about the GRIB format the reader is referred to Dey (1998). The only information contained in a GRIB file with which we are concerned is the wind information.

Wind speed and direction are defined at a number of times on a grid of points covering an area of the earth’s surface. The spatial grid is assumed to be equally spaced in latitude and longitude, although GRIB files support other grid types as well. In order to obtain the wind at a certain position and time it is necessary to interpolate between the known values.

To perform the interpolation in space, we treat the grid of known wind vectors as a finite element mesh. The vertices of the grid constitute nodes, while rectangles with these nodes at the corners constitute bilinear elements. Bilinear interpolation was chosen for ease and speed of computation. An example of a bilinear element is shown in Figure 2.11.

Figure 2.11 Spatial interpolation inside a wind element. Bold arrows are known wind vectors and the others are interpolated.

The interpolation is performed independently on the components of the wind vectors. It is not clear whether it is best to interpolate the speed and direction or latitude and
longitudinal components of the wind vectors. Figure 2.12 shows both cases for a onedimensional interpolation between two wind vectors of equal magnitude separated by almost 180°. Case (a), interpolating speed and direction components, better approximates the case of the wind swinging from one direction to another without changing in speed while case (b), interpolating latitude and longitude components, better approximates a wind ‘hole’ between the opposing wind vectors. It is common practice in meteorology to use the latitude and longitude components (White, N., personal communication with author, April 27, 1998). In the following formulae we will use the speed and direction components, however both options are available in the stochastic routing software.

![Figure 2.12 Interpolating (a) speed and direction components and (b) latitude and longitude components.](image)

The wind at a point within an element with known wind vectors $w_1$, $w_2$, $w_3$, and $w_4$ at corner positions $x_1$, $x_2$, $x_3$, and $x_4$ respectively is given by

$$w_r(x) = \sum_{i=1}^{4} \psi_i(x_1, x_2)w_i^r$$

$$w_\theta(x) = \sum_{i=1}^{4} \psi_i(x_1, x_2)w_i^\theta,$$

(2.13)

where $\psi_i, i = 1, 2, 3, 4$ are the bilinear basis functions,

$$\psi_1(x_1, x_2) = (1 - x_1)(1 - x_2)$$

$$\psi_2(x_1, x_2) = (1 - x_1)x_2$$

$$\psi_3(x_1, x_2) = x_1x_2$$

$$\psi_4(x_1, x_2) = x_1(1 - x_2)$$
and \((\xi_1, \xi_2)\) are the element coordinates, given by

\[
\xi_1 = \frac{x_{lon} - x_{lon}^1}{x_{lon}^2 - x_{lon}^1}, \\
\xi_2 = \frac{x_{lon}^2 - x_{lon}}{x_{lon}^2 - x_{lon}^1}.
\]

Care needs to be taken when interpolating the direction of these vectors when their directions differ by more than 180°. Two one-dimensional examples of this case are shown in Figure 2.13. Interpolating between \(\theta_1\) and \(\theta_2\) yields an angle around 180° rather than an angle around 0° and interpolating between \(\theta_1\) and \(\theta_2\) yields an angle around 0° rather than an angle around 180°. This can be avoided by ensuring that no pair of angles involved in the interpolation differ by more than 180°. This can be achieved by adding 360° to or subtracting 360° from selected angles.

![Figure 2.13 Examples of interpolating angles more than 180° apart.](image)

If the wind is required at a time at which it is not defined in the GRIB file then it is necessary to interpolate in time. It is not immediately obvious how to go about this because accurate prediction of the way in which weather evolves in time requires a very complex model and this is beyond the scope of this work. In the absence of any better method, wind is interpolated linearly in time, that is

\[
\begin{align*}
    w_r(t) &= (1 - \xi) w_r(t_1) + \xi w_r(t_2), \\
    w_\theta(t) &= (1 - \xi) w_\theta(t_1) + \xi w_\theta(t_2),
\end{align*}
\]  

(2.14)
where \( \xi_i = \frac{(t - t_i)}{(t_2 - t_1)} \) and \( t_1 \leq t \) and \( t_2 \geq t \) are times at which the wind is defined in the GRIB file. Combining (2.13) and (2.14) gives

\[
w_r(x, t) = (1 - \xi_i) \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) w'_r(t_i) + \xi_i \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) w'_r(t_2)
\]

\[
w_\theta(x, t) = (1 - \xi_i) \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) w'_\theta(t_i) + \xi_i \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) w'_\theta(t_2)
\]

### 2.3.2 Current

Information about current is not obtained from GRIB files, but from a separate current file. Because current is not as difficult to predict as wind, current predictions are prepared in advance for the period and area of interest. Like wind information, current information is stored as a series of grids, one for each of a number of times. The interpolation of current follows the same form as that of wind, with the exception that the latitude and longitude components are interpolated rather than the speed and direction components. This gives the interpolation formula

\[
c_{lon}(x, t) = (1 - \xi_i) \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) c'_{lon}(t_i) + \xi_i \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) c'_{lon}(t_2)
\]

\[
c_{lat}(x, t) = (1 - \xi_i) \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) c'_{lat}(t_i) + \xi_i \sum_{i=1}^{4} \psi_i(\xi_i, \xi_2) c'_{lat}(t_2)
\]

Because no angles are being interpolated, the considerations when interpolating angles more that 180° apart do not apply.

### 2.4 Summary

In this chapter we have defined the coordinate system we will use to describe points on the surface of the earth and formulae for calculation of distances and angles in this coordinate system. Furthermore, the model used for predicting a yacht’s velocity was described, yielding speed in a given direction as a function of wind and current for a given yacht.

The next chapter introduces a discretisation of the problem domain and presents a method for determining optimal paths when the weather is known in advance.
3. Deterministic Routing

In the previous chapter the model used for determining the speed of a yacht when sailing in a given direction for given wind and current was described. This chapter introduces a discretisation of the problem domain and describes a method for calculating an optimal path on this discretisation when the wind and current are known in advance. Ultimately, this chapter serves as background for the stochastic routing algorithm described in Chapter 5; it is simpler to introduce the dynamic programming method used for both the deterministic and stochastic problems without the complication of uncertain weather.

In deterministic routing we seek the route to sail from the start to the finish that minimises the time taken, assuming perfect knowledge of the weather. The parameters that define the problem are start and finish positions \( x_{\text{start}} \) and \( x_{\text{finish}} \), start time \( t_{\text{start}} \), and wind \( w(x, t) \) and current \( c(x, t) \).

### 3.1 Network Definition

The problem of finding the route that takes the minimum time can be viewed as a shortest-path problem. The method we consider is a common way of solving such problems and involves discretising the solution space into a network consisting of a set of discrete positions through which the optimal route may pass.

In network terminology, the points through which a route may pass are known as nodes. An arc represents sailing from one node to another. The set of nodes and all possible arcs forms a graph. A path is a sequence of nodes joined by arcs. These concepts are illustrated in Figure 3.1. This example will be used throughout the rest of the discussion of deterministic methods.
The nodes are identified by number and the set of \( n \) nodes is

\[
N = \{1,2,\ldots,n\}.
\]

The set of \( m \) arcs is

\[
A = \{(i_1,j_1),(i_2,j_2),\ldots,(i_m,j_m)\},
\]

where \((i_1,j_1)\) is the arc from node \( i_1 \) to node \( j_1 \). The arcs are directed, meaning that \((i_1,j_1)\) is different to \((j_1,i_1)\).

In Figure 3.1, \( N = \{1,2,\ldots,5\} \) and \( A = \{(1,2),(1,3),(1,4),(2,5),(3,5),(4,5)\} \).

A set of nodes \( N \) and a set of arcs \( A \) define a graph \( G \), where

\[
G = (N,A).
\]

Because the arcs are directed, this is strictly a digraph (directed-graph), but because all graphs discussed are directed we will refer to it simply as a graph with no ambiguity.

In our shortest path problem the nodes identify positions on the ocean. We will call the position identified by node \( i \) \( x_i \), and thus we can define the travel time for the arc \((i,j)\) between two nodes, \( i \) and \( j \), starting at time \( t \) as

\[
c_{arc}(i,j,t) = c_{seg}(x_i,x_j,t),
\]

\[ (3.1) \]
where \(c_{\text{seg}}(x_i, x_j, t)\) is the time taken to sail the great circle joining \(x_i\) and \(x_j\) starting at time \(t\), defined in (2.3).

The solution to our problem takes the form of a path. A path is a graph of the form

\[
P = (N(P), A(P)),
\]

\[
N(P) = \{n_0, n_1, \ldots, n_l\},
\]

\[
A(P) = \{(n_0, n_1), (n_1, n_2), \ldots, (n_{l-1}, n_l)\}
\]

that is a sequence of nodes joined by arcs. If we assume that the arcs between successive nodes in the path are unique then we can define the path by only the sequence of nodes. In this case the order of the nodes is important and we will refer to \(P[i]\), meaning the \(i^{th}\) node along the path. Note that this could also be written \(N(P)[i]\).

Also, we will write \(|P|\), meaning the number of nodes in the path, where \(|P| = |N(P)|\).

In the example of Figure 3.1, \(N(P) = \{1, 4, 5\}\) and \(A(P) = \{(1, 4), (4, 5)\}\).

In order to use a network approach to solving the shortest path problem, the space in which a solution is desired must be discretised, yielding a graph. The solution to our problem is a path within this graph that starts at the start position \(x_{\text{start}}\) and finishes at the finish position \(x_{\text{finish}}\). We will call the set of such paths \(P_{x_{\text{start}}x_{\text{finish}}}\).

The problem is to find the path \(P^* \in P_{x_{\text{start}}x_{\text{finish}}}\) with the least travel time. The travel time for a path is given by summing the travel times of each of its arcs.

Thus the solution to the problem, starting at position \(x_{\text{start}}\) and time \(t_{\text{start}}\) and finishing at position \(x_{\text{finish}}\) is

\[
P^* = \operatorname{arg\,min}_{P \in P_{x_{\text{start}}x_{\text{finish}}}} c_{\text{seg}}(P, t)\]

(3.2)

The following definitions will be used in the analysis in the next section. For more depth the reader is referred to Gondran and Minoux (1984).
A node $j$ is a successor of a node $i$ if there exists an arc $(i, j)$. If this is the case then the node $i$ is termed a predecessor of the node $j$. We define the set of successors, $\Gamma_i$, of a node $i \in N(G)$ as $\Gamma_i = \{j \in N(G): (i, j) \in A(G)\}$.

Similarly, the set of predecessors, $\Gamma_j^{-1}$, of a node $i \in N(G)$ is $\Gamma_j^{-1} = \{i \in N(G): (i, j) \in N(G)\}$.

A $p$-graph is a graph in which there are at most $p$ arcs between any pair of nodes. The graphs we will consider are all 1-graphs. This can be justified by considering the assumption that arcs are always sailed as fast as possible using great circles. A 1-graph can be represented by its set of nodes and set of successors or predecessors, that is we can write $G_i = (N, \Gamma_i)$ or $G_i^{-1} = (N, \Gamma_i^{-1})$ where $\Gamma$ and $\Gamma^{-1}$ are transformations that map each element of $N(G)$ to a subset of $N(G)$. We call these transformations multivalued mappings. The set of arcs $A$ can be derived from the multivalued mapping $\Gamma$ for a 1-graph by $A = \{(i, j): i \in N, j \in \Gamma_i\}$, and similarly from the multivalued mapping $\Gamma^{-1}$ by $A = \{(i, j): j \in N, i \in \Gamma_j^{-1}\}$.

A chain of length $q$ is a sequence of $q$ arcs $L=\{(a_1^1, a_1^2), (a_2^1, a_2^2), \ldots, (a_q^1, a_q^2)\}$ where each arc $(a_i^1, a_i^2)$ has one node in common with arcs $(a_{i+1}^1, a_{i+1}^2)$ and $(a_{i-1}^1, a_{i-1}^2)$ for $i = 2,3,\ldots,q-1$. Note that a path is a chain in which all the arcs are oriented in the same direction. The endpoints of a chain $L$ are the nodes in the arcs $(a_1^1, a_1^2)$ and $(a_q^1, a_q^2)$ that are not also in the arcs $(a_2^1, a_2^2)$ and $(a_{q-1}^1, a_{q-1}^2)$ respectively. A cycle is a chain whose endpoints coincide.

### 3.2 Dynamic Programming

Dynamic programming is a solution approach that divides the solution into a sequence of decisions. In the shortest path problem the solution is a sequence of nodes, starting at the node at the start position $x_{\text{start}}$ and finishing at the node at the finish position $x_{\text{finish}}$. At each node, the decision to be made is that of which node to move to next in the sequence, and in what state to arrive. The state of a node is some information that depends on the sequence of decisions taken to reach that node. For the shortest path
problem, the state of a node $i$ is the time at which we wish to depart that node. We can also, without ambiguity, refer to a state as being a node-time pair. If we assume that we will always sail as fast as possible then the state in which we reach node $j$ from node $i$ is determined by the sum of the time at which we depart node $i$ (the state of node $i$) and the time taken to sail the arc $(i, j)$. Note that this time, $c_{arc}(i, j, t_i)$, also depends on the state of node $i$, where $t_i$ is this state.

The three possible paths in our simple example are shown in Figure 3.2 in which the state of the nodes is plotted on the vertical axis. It can be seen that the finish node, $n_{finish}$, is reached at three different times. The optimal solution is the path that ends on the finish node at the earliest time, and is shown using bold lines. This sort of diagram is useful in describing the stochastic solution method in Chapter 5.

![Figure 3.2](image)

**Figure 3.2** Three paths through a simple graph from $n_{start}$ to $n_{finish}$. Each relevant node-time pair $(i, t)$ is labelled. Note that the optimal path is the one that reaches node 5 ($n_{finish}$) at the earliest time, $t = 3$ in this case; this path is shown in bold.

The fundamental assumption of dynamic programming is that the optimal sequence of decisions from any node-time pair depends only on the node and state information. For the shortest-path problem, this means that the optimal path from any node depends only on that node’s position and the time at which we leave. Also, in order to describe the optimal path from any node for a given state, it is only necessary to store the next node-
time pair, because the optimal path from that node-time pair is defined by its optimal successor and so on until the finish node, \( n_{\text{finish}} \), is reached. This leads to a recursive definition of the optimal sequence of decisions, which we will call the *forward-looking* recursion:

\[
\begin{align*}
    f^*(i,t) &= \begin{cases} 
        0, & \text{if } i = n_{\text{finish}} \\
        \min_{j \in \Gamma} \left[ c_{ij}(i,t) + f^*(j,t + c_{ij}(i,j,t)) \right], & \text{otherwise}
    \end{cases}, \\
    j^*(i,t) &= \arg \min_{j \in \Gamma} \left[ c_{ij}(i,t) + f^*(j,t + c_{ij}(i,j,t)) \right], \quad i \neq n_{\text{finish}}
\end{align*}
\]

(3.3)

where \( f^*(i,t) \) is the time taken for the optimal sequence of decisions from the node-time pair \((i,t)\) to the finish node and \( j^*(i,t) \) is the successor of \( i \) on the optimal path when in state \( t \). In other words, the quantity being minimised at each node-time pair \((i,t)\) is the time to reach the finish node, \( n_{\text{finish}} \). Note that at least one node is at the finish position \( x_{\text{finish}} \), but that there is only one finish node.

The optimal time taken to reach \( n_{\text{finish}} \), which is the quantity we are ultimately minimising, is given by \( f^*(n_{\text{start}},t_{\text{start}}) \) and the path that yields this time can be obtained by following the optimal successors. An algorithm for this is shown in Algorithm 1.
Algorithm 1 Extraction of the optimal path $P^*$ from node $n_{\text{start}}$ at time $t_{\text{start}}$ to node $n_{\text{finish}}$ from the optimal successor information.

(1) $P^* = (N(P^*), A(P^*)) \leftarrow (\emptyset, \emptyset)$

(2) $i \leftarrow n_{\text{start}}$; $t \leftarrow t_{\text{start}}$

(3) $N(P^*) \leftarrow N(P^*) \cup \{i\}$

(4) If $i \neq n_{\text{finish}}$ then
   a) $j \leftarrow j^*(i, t)$
   b) $A(P^*) \leftarrow A(P^*) \cup \{(i, j)\}$
   c) $t \leftarrow t + c_{\text{arc}}(i, j, t)$
   d) $i \leftarrow j$
   e) Go to (3)

(5) Else Stop

We call (3.3) the forward-looking recursion because the optimal successor and optimal time to go from each node are found by examining that node’s successors. Because a node’s optimal successor and optimal time to go are functions of the optimal times to go of all of that node’s successors, they can not be calculated until all of its successor’s optimal times to go have been calculated. We will use the word visit to denote the act of calculating the optimal time to go and optimal successor from a node-time pair, and the word label to denote the act of finishing that calculation. Thus, the deterministic problem is solved by labelling the node-time pair $(n_{\text{start}}, t_{\text{start}})$, but this node-time pair can not be labelled until all of its successors have been labelled. This means that $(n_{\text{start}}, t_{\text{start}})$ is the last node-time pair to be labelled, but the first to be visited; we could also refer to (3.3) as forward-visiting, but backward-labelling. The order in which the nodes can be labelled can be calculated in advance.

The node-time pair $(i, t)$ can be labelled if $f^*(j, t + c_{\text{arc}}(i, j, t))$ is known for all $j \in \Gamma_i$. We know that $f^*(n_{\text{finish}}, t) = 0, \forall t$, so all nodes whose successor set consists only of $n_{\text{finish}}$, that is $i : \Gamma_i = \{n_{\text{finish}}\}$, can be labelled simply by
Similarly, we can then calculate \( f^*(j,t) \) for, and thus label, all nodes whose successor set contains only those nodes we have already labelled, that is \( j : \Gamma_j = \{ i : \Gamma_i = \{ n_{\text{finish}} \} \} \).

The only difference is that a minimisation is necessary in this case, that is
\[
f^*(j,t) = \min_{i : \Gamma_i = \{ n_{\text{finish}} \}} \left[ c_{\text{arc}}(j,i,t) + f^*(i,t + c_{\text{arc}}(j,i,t)) \right], \quad \forall j : \Gamma_j = \{ i : \Gamma_i = \{ n_{\text{finish}} \} \}.
\]

Then the nodes whose successors are these nodes \( j \) can be labelled and so on until we have calculated \( f^*(n_{\text{start}}, t_{\text{start}}) \). If the graph contains no cycles then we need only make one pass through the nodes as long as we label the nodes in order of their rank.

The rank of a node \( i \) in a graph \( G \) is the number of arcs in the longest path (in terms of number of arcs) between node \( n_{\text{start}} \) and node \( i \) and is written \( r(i) \). We will define the set \( R_k = \{ i \in N(G) : r(i) = k \} \) and call it stage \( k \). The start node, \( n_{\text{start}} \), has a rank of zero and every node that can be reached from the start node has a rank greater than zero. The set of stages will be denoted by \( R \) and the greatest rank by \( r_{\text{max}} = \max_{i \in N(G)} r(i) \). Note that \( r(n_{\text{finish}}) \) is not necessarily equal to \( r_{\text{max}} \), but that any nodes with rank greater than \( r(n_{\text{finish}}) \) will never form a part of the optimal path. This is because it is not possible to reach a node \( j \) from a node \( i \) if \( r(j) < r(i) \), and the optimal path must end on \( n_{\text{finish}} \). It follows that every node in the optimal path has rank less than or equal to \( r(n_{\text{finish}}) \). Furthermore, there is only one node in the optimal path with rank equal to \( r(n_{\text{finish}}) \), that is \( n_{\text{finish}} \) itself.

The example in Figure 3.3 shows the rank of nodes in a graph. In this case \( R = \{ R_0, R_1, R_2, R_3 \} \), where \( R_0 = \{ n_{\text{start}} \} \), \( R_1 = \{ 2, 3, 4 \} \), \( R_2 = \{ 5, 6 \} \), and \( R_3 = \{ n_{\text{finish}} \} \).
We know \( f^* \) for the single node with rank \( r(n_{\text{finish}}) \) (that is \( f^*(n_{\text{finish}},t) = 0 \)) and we know that \( n_{\text{finish}} \) can be reached from all nodes with rank \( r(n_{\text{finish}}) - 1 \) in one arc. We also know that all nodes with rank \( r(n_{\text{finish}}) - 1 \) only have as successors nodes with rank \( r(n_{\text{finish}}) \) or higher. Because we are not considering nodes with rank higher than \( r(n_{\text{finish}}) \), we have enough information to calculate \( f^* \) for all nodes with rank \( r(n_{\text{finish}}) - 1 \). By induction, nodes can be labelled in order of decreasing rank.

The assumption necessary for an ordering based on rank to be the same as one based on successor sets is that all of a node’s successors have rank exactly one greater than that node. The example in Figure 3.3 violates this assumption because node 4, which has a rank of 1, has node 7, which has rank 3, in its successor set. Because successor sets are generated from rank information in practice, this assumption will be valid for all graphs we will consider.

Thus, the set of nodes \( i : \Gamma_i = \{ n_{\text{finish}} \} \) can be replaced with \( i : r(i) = r(n_{\text{finish}}) - 1 \), \( j : \Gamma_j = \{ i : \Gamma_i = \{ n_{\text{finish}} \} \} \) with \( j : r(j) = r(n_{\text{finish}}) - 2 \), and so on.

However, we do not know in advance which states at each node will be visited, so, although we know the order in which the nodes must be labelled, we don’t know which states at those nodes need to be labelled.
The simplest way, conceptually, of dealing with this problem is to effectively ignore it by labelling each node at all times. In this way there is a well-defined order of evaluation that depends only on the graph, and not on any other problem variables, and so can be determined before anything is known about the problem. However, because each node is labelled at all times in a continuous, infinite set, this approach is not practicable.

We will consider two ways to deal with this problem. One is to literally recurse, reaching forward from each visited node until all nodes are labelled, and the other is to consider the problem from the other end, so to speak, and find optimal predecessors rather than optimal successors. The latter is preferable for the deterministic problem, but can not be used for the stochastic problem because of an asymmetry in the information flow that will be discussed in Chapter 5.

3.2.1 Recursion
If we only consider states such that \[ t = t_{\text{start}} + k\Delta t, \] where \( k = 0,1,... \) and \( \Delta t \) is a constant, then it is possible that a node will be reached in the same state from more than one of its predecessors. An example of this is shown in Figure 3.4. Node 5 is reached at time 7 from both node 2 and node 4. It is not necessary to label the node-time pair \((5,7)\) twice, thus reducing the number of calculations required at node 5, and also reducing the number of time states at which the successors of node 5 are reached.
3.2 Dynamic Programming

Figure 3.4 Example showing a node reached at the same time from multiple predecessors when the time state has been discretised. Note that $\Delta t = 1$ in this case.

This is still an inefficient approach for the deterministic problem, because more node-time pairs are labelled than is necessary, but it is shown to be necessary for the stochastic problem.

3.2.2 Backward-looking

The forward-looking recursion defines an order in which the nodes must be labelled (that of decreasing rank), but it is not known in advance which states at these nodes need to be labelled. For the deterministic problem, however, it is possible to define a backward-looking recursion for which both the order of evaluation and the states at which evaluation must occur can be determined.

The key to developing a backward-looking technique is to realise that, in the deterministic problem, when traversing the optimal path, each node in the path will be reached at the earliest possible time.

**Lemma** – Each node on the optimal path is reached at the earliest possible time.

**Proof** – This is obviously true for $n_{\text{finish}}$, and must therefore also be true for the node immediately preceding it. This follows from the assumption that the travel times for an
arc at times $t_1$ and $t_2$ where $t_1 < t_2$ satisfy $t_1 + c_{arc}(i, j, t_1) \leq t_2 + c_{arc}(i, j, t_2)$, that is it is impossible to leave later but arrive earlier. The result follows by induction.$\square$

What this means is that, if we know the earliest time at which a node can be reached, it is only necessary to label it at that time. Thus, it is desirable to calculate the earliest time at which each node can be reached. We will define $t(j)$ to be the earliest time at which node $j$ can be reached. This leads to the backward-looking recursion:

$$
t(j) = \begin{cases} 
t_{start}, & j = n_{start} 
\min_{i \in \Gamma^{-1}_j} \{c_{arc}(i, j, t(i)) + t(i)\}, & \text{otherwise} 
\end{cases} \tag{3.4}
$$

where $t(j)$ is the time at which node $j$ is reached on the optimal path, $\Gamma^{-1}_j$ is the set of possible predecessors from $j$, $p(j)$ is the predecessor of $j$ on the optimal path, and $n_{start}$ is the first node on the optimal path. The set $\Gamma^{-1}_j$ can be defined in terms of the successor set $\Gamma_j$ by $\Gamma^{-1}_j = \{i \in N(G): j \in \Gamma_i\}$.

We will refer to the pair $(t(j), p(j))$ as the label of node $j$, and to the act of calculating these numbers as labelling node $j$. Figure 3.5 shows predecessor sets and optimal predecessors for our simple example.
Similarly to the forward case, node \( j \) can be labelled without recursion if \( t(i) \) is known for all \( i \in \Gamma_j^{-1} \). It is therefore desirable to find an order in which to label the nodes such that (3.4) requires only information about nodes that have already been labelled. As in the forward case, this order can be defined in terms of the rank of the nodes. All predecessors of a node must have a lower rank than that node, so it is sufficient to label nodes in order of increasing rank. Labelling nodes in this order can be done using Algorithm 2, which is Ford’s algorithm (Ford, 1956) for a graph with no cycles as described by Gondran and Minoux (1984).

**Algorithm 2** Single iteration Ford’s algorithm for nodes grouped by rank.

1. \( t(n_{\text{start}}) \leftarrow t_{\text{start}} \), \( t(i) \leftarrow \infty, i \in N(G), i \neq n_{\text{start}} \), \( p(i) \leftarrow -1, i \in N(G) \)

2. For each stage \( k = 1, 2, \ldots, r_{\text{max}} \)
   a) For each node \( j \in R_k \)
      i) For each node \( i \in \Gamma_j^{-1} \)
         (a) If \( t(i) + c_{\text{arc}}(i, j, t(i)) < t(j) \)
             (i) \( t(j) \leftarrow t(i) + c_{\text{arc}}(i, j, t(i)) \)
             (ii) \( p(j) \leftarrow i \)
Because it is more convenient to define the graph using successor sets, our implementation of this algorithm uses successor sets rather than predecessor sets and proceeds as follows:

**Algorithm 3** Single iteration Ford’s algorithm using successor sets for nodes grouped by rank.

1. \( t(n_{\text{start}}) \leftarrow t_{\text{start}} \); \( t(i) \leftarrow \infty, i \in N(G), i \neq n_{\text{start}} \); \( p(i) \leftarrow -1, i \in N(G) \)

2. For each stage \( k = 0,1,\ldots,r_{\text{max}}-1 \)
   a) For each node \( i \in R_k \)
   i) For each node \( j \in \Gamma_i \)
      a) Calculate \( c_{\text{arc}}(i, j, t(i)) \)
      b) If \( t(i) + c_{\text{arc}}(i, j, t(i)) < t(j) \)
         i) \( t(j) \leftarrow t(i) + c_{\text{arc}}(i, j, t(i)) \)
         ii) \( p(j) \leftarrow i \)

The major calculation required in this algorithm is that of the arc cost in (2)a)i)(a). This calculation is carried out once for each successor of each node of each rank except \( n_{\text{finish}} \). If \( C \) is the number of arc cost calculations required, then

\[
C = \sum_{r=0}^{r(n_{\text{finish}})-1} \sum_{i \in R_k} |\Gamma_i|.
\]

If we assume that there are \( N \) nodes in each stage then \(|\Gamma_i| = N, \forall i \in N(G), i \neq n_{\text{finish}}\) and

\[
C = \sum_{r=0}^{r(n_{\text{finish}})-1} \sum_{i \in R_k} |\Gamma_i| = \sum_{r=0}^{r(n_{\text{finish}})-2} N + \sum_{i \in R_k, i \neq n_{\text{finish}}} 1 = (r(n_{\text{finish}})-1)N^2 + N
\]

If we let \( M = r(n_{\text{finish}}) \), then the complexity of this algorithm is thus \( O(C) \), where
It will be shown in §3.3.2 that it is necessary to increase the number of nodes in each stage linearly with the number of stages in order to maintain acceptable angle resolution, so the complexity of the algorithm is effectively $O(M^3)$ for a fixed problem domain.

Once each node has been labelled with its optimal arrival time and predecessor, the optimal path $P^*$ from node $n_{\text{start}}$ to the node $n_{\text{finish}}$ (or any other node) can be extracted by tracing the optimal predecessors back from $n_{\text{finish}}$.

**Algorithm 4** Extraction of the optimal path $P^*$ from node $n_{\text{start}}$ to node $n_{\text{finish}}$ from the optimal arrival time and predecessor labels.

1. $P^* = \left( N(P^*), A(P^*) \right) \leftarrow (\emptyset, \emptyset)$
2. $j \leftarrow n_{\text{finish}}$
3. do until $j = -1$
   a) $N(P^*) \leftarrow \{j\} \cup N(P^*)$
   b) $i \leftarrow p(j)$
   c) if $i \neq -1$ then $A(P^*) \leftarrow \{i, j\} \cup A(P^*)$
   d) $j \leftarrow i$

3.3 Graph generation

If the solution generated by this method is to be a good approximation to the continuous optimal solution, the graph used must have two properties. (a) it must cover the area through which the true solution passes and (b) it must be fine enough to support a path close to the true solution.

3.3.1 Graph Domain

We consider a technique for generating the discretisation of the problem domain that we require in order to form the network used in the dynamic programming solution method.
This technique takes as input a description of a corridor through which the yacht is permitted to sail and a node and stage spacing. This description takes the form of a sequence of pairs of positions. Each position of each pair defines one of the two edges of the corridor. Figure 3.6 shows an example of such a permissible region. In this case there are five pairs of positions. A dashed line joins each pair. Note that both points in the first and last pairs are identical; this is because the start and finish points must be uniquely defined.

![Figure 3.6 Permissible sailing region defined by a boundary.](image)

Once such a description of the domain is known, a graph is generated in two phases. First the positions of the stages are determined, and then nodes are placed on the stages. The stages are chosen to be straight lines such that the maximum distance between adjacent stages is not greater than the desired stage spacing. An example of stage position choice is shown in Figure 3.7.

![Figure 3.7 Lines on which nodes are generated. The nodes on each line form a stage.](image)

Nodes are then placed on each of these lines such that all pairs of adjacent nodes within each stage are not further apart than the desired node spacing.
3.3.2 Graph Resolution

Recall that the routing problem can be treated as an optimal control problem where the control variable is the direction of travel of the yacht, $\theta$. Although we have formulated the problem in such a way that the decisions made are those of position, decisions of direction remain the more natural way to consider the problem. This is due to the way in which a yacht implements a route; the only control a yacht has over where it goes is obtained by setting the heading of the yacht. While the heading of the yacht is not, in general, the same as the direction in which it travels, they share enough properties for them to be interchangeable for the purposes of this discussion of graph generation.

Because direction is the natural control variable, it is important that the possible choices of direction are sufficiently close together; the closer together these possible choices are, the less error there will be in the solution. Thus, the actual maximum distance apart that can be considered sufficient depends on how much accuracy is required. We will characterise the spacing of choices of direction by the maximum difference in direction encountered when considering adjacent successors to each node. This is illustrated in Figure 3.8.

![Figure 3.8](image)

**Figure 3.8** Excerpt from a graph showing possible successors from nodes 1 to 5 and differences in direction between the possible successors from node 3.
The possible successors from a single node, node 3, are shown using bold lines, and the differences in direction between adjacent successors are labelled $\beta_i^{j,k}$, where $\beta_i^{j,k}$ is the difference in direction, when starting from node $i$, between going to node $j$ and going to node $k$. The maximum of these differences is

$$\beta_i^{\text{max}} = \max_{m \in [i, I]} \beta_i^{[m], [m+1]}.$$ 

and the maximum for the entire graph is

$$\beta^{\text{max}} = \max_{i \in V(G)} \beta_i^{\text{max}}.$$ 

We will use this maximum difference in direction to characterise the angle resolution of the graph. In the case when stages are straight lines of equally spaced nodes, this value can be easily calculated. Figure 3.9 shows a node, A, and two adjacent successors, B and C. In this example, the successor stage is a distance $\Delta_{\text{stage}}$ away from the node A, and the nodes in the successor stage are equally spaced at a distance of $\Delta_{\text{node}}$.

![Figure 3.9](image)

*Figure 3.9 Calculation of angle resolution for equally spaced nodes in straight-line stages when nodes in adjacent stages are aligned.*

The angle $\beta$ (which could be written $\hat{BAC}$) is given by

$$\beta = \tan^{-1} \frac{\Delta_{\text{node}}}{\Delta_{\text{stage}}}.$$ 

It is clear that this angle is the largest angle between adjacent successors of node A, that is $\beta_A^{\text{max}} = \beta$. If all other stages have the same node spacing $\Delta_{\text{node}}$ and stage spacing $\Delta_{\text{stage}}$, then this value is the largest direction difference for the entire graph, that is $\beta^{\text{max}} = \beta_A^{\text{max}} = \beta$. 
Now, consider changing the stage spacing to $\lambda^{\text{stage}}$. In order to maintain the same angle resolution, it is necessary to change the node spacing also. We will call the node spacing required to maintain the angle resolution $\Delta_1^{\text{node}}$. This satisfies

$$\tan^{-1}\left(\frac{\Delta_1^{\text{node}}}{\lambda^{\text{stage}}\Delta_1^{\text{stage}}}\right) = \tan^{-1}\left(\frac{\Delta^{\text{node}}}{\Delta^{\text{stage}}}\right).$$

Thus, if the stage spacing is reduced ($\lambda < 1$), the node spacing must be reduced by the same factor to ensure that the angle resolution is not worsened. A similar argument applies for the case when the node spacing is increased. If we have a maximum acceptable value of the angle resolution, $\beta^{\text{max}}$, then we can formulate a constraint on acceptable values of the node and stage spacing:

$$\beta^{\text{max}} = \tan^{-1}\left(\frac{\Delta^{\text{node}}}{\Delta^{\text{stage}}}\right).$$

Because $0 < \beta^{\text{max}} < 90^\circ$, we can take the tangent of both sides and maintain the inequality, giving

$$\tan \beta^{\text{max}} \geq \frac{\Delta^{\text{node}}}{\Delta^{\text{stage}}}, \quad (3.5)$$

which we will call the angle resolution constraint.

The optimal paths, as produced by the deterministic routing algorithm for different node spacings and constant stage spacing of 100NM, are shown in Figure 3.10. The weather used in this example is a forecast sent to yachts competing in The Whitbread Round The World Race in January 1998.
Figure 3.10 Optimal paths for four different node spacings with constant stage spacing, where the start point is to the left and the destination is to the right.

The graph shown in this screenshot was generated with the coarsest of the four node spacings; the optimal path for this graph is shown in blue. The final line segment of this path is shown as a dashed line; this indicates that the penultimate point on this path is reached at a time after the final weather forecast, so a default wind was used. The remaining three paths were calculated using graphs in which the node spacing was progressively halved. Note that all of these remaining paths do not exceed the time covered by the weather forecast. The node spacings, angle resolutions and colours, are shown in Table 3.1 for each graph. This table also shows the optimal travel times; it can be seen that finer graphs result in shorter travel time solutions. Note that, because the node spacing is always a multiple of the smallest node spacing, the optimal route under the larger node spacings remain feasible solutions under the smaller node spacings. Thus it is not possible for the optimal route time to increase when reducing the node spacing.
### Table 3.1 Node spacings and travel times for the four optimal paths shown in Figure 3.10.

<table>
<thead>
<tr>
<th>Colour</th>
<th>Node spacing $\Delta^{node}$ (NM)</th>
<th>Angle resolution $\beta^{max}$</th>
<th>Optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>15</td>
<td>8.53°</td>
<td>4d 23:19:15 (\approx) 119.32h</td>
</tr>
<tr>
<td>Green</td>
<td>30</td>
<td>16.70°</td>
<td>4d 23:26:23 (\approx) 119.44h</td>
</tr>
<tr>
<td>Yellow</td>
<td>60</td>
<td>30.96°</td>
<td>5d 03:45:04 (\approx) 123.75h</td>
</tr>
<tr>
<td>Blue</td>
<td>120</td>
<td>50.19°</td>
<td>5d 07:06:00 = 127.1 h</td>
</tr>
</tbody>
</table>

It can be seen that as the node spacing is increased, and the angle resolution is thus also increased, the optimal path becomes less smooth, that is it contains sharper changes in direction. This is clearly because of the coarser angle resolution; there are fewer possible choices of direction from each node and those choices are further apart. This example illustrates the adverse effects of increasing the node spacing without proportionally increasing the stage spacing to maintain the same angle resolution. The next example shows the effect of decreasing the stage spacing without decreasing the node spacing. Figure 3.11 shows the optimal paths for four graphs, all with a node spacing of 15NM, with different stage spacings.
Figure 3.11 Optimal paths for four different stage spacings with constant node spacing, where the start point is to the left and the destination is to the right.

The four different stage spacings in this example cover the same angle resolutions as the different node spacings in the previous example. The stage spacings and corresponding angle resolutions and optimal route times are shown in Table 3.2.
3.4 Summary

Table 3.2 Stage spacings and travel times for the four optimal paths shown in Figure 3.11.

<table>
<thead>
<tr>
<th>Colour</th>
<th>Stage spacing $\Delta^{\text{stage}}$ (NM)</th>
<th>Angle resolution $\beta^\text{max}$</th>
<th>Optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>100</td>
<td>8.53º</td>
<td>4d 23:19:15 ≈ 119.32h</td>
</tr>
<tr>
<td>Green</td>
<td>50</td>
<td>16.70º</td>
<td>4d 23:54:22 ≈ 119.91h</td>
</tr>
<tr>
<td>Yellow</td>
<td>25</td>
<td>30.96º</td>
<td>5d 01:24:28 ≈ 121.41h</td>
</tr>
<tr>
<td>Blue</td>
<td>12.5</td>
<td>50.19º</td>
<td>5d 08:27:34 ≈ 128.46h</td>
</tr>
</tbody>
</table>

A pattern similar to that observed for increasing the node spacing can be seen, that is as the angle resolution is increased, the optimal time is increased and the optimal path contains sharper changes in direction. The latter is to be expected simply because there are fewer choices of direction available. The former, however, is evidence that increasing the angle spacing is detrimental to the quality of the solution.

3.4 Summary

In this chapter, we have described the discretisation of the problem domain and a method for determining the optimal path when the wind and current are known in advance. A method for generating this discretisation was presented, and the issue of angle resolution was discussed.

The next chapter describes a technique for generating isochrones from information resulting from the solution method. This material is peripheral to the central theoretical argument of the thesis and may be skipped without jeopardising the reader’s understanding of the remainder of this work.

In the following chapter, we develop a technique for routing when the wind is uncertain. A model for describing the stochastic nature of the wind is described. It is shown that an approach similar to the deterministic method is possible, but that the dynamic programming must be left in the forward direction rather than be reversed as was
effective for the deterministic case. Methods for compensating for the resultant increase in time-complexity are investigated.
4. Isochrones

Most existing weather routing software produces solutions using isochrones, or lines of equal time. An isochrone represents the maximum distance it is possible to sail from the starting location in a given time. If the wind is uniform in space and time then all isochrones will be the same shape as the polar, but in general this is not the case. The method by which isochrones are generated by commercial software packages is not freely available information.

Because isochrones give valuable information to navigators, a technique was developed to generate isochrones from the solution obtained using the dynamic programming algorithm described in the previous chapter.

4.1 Isochrone Generation

In order for the optimal path to be determined using this algorithm, each node \( j \) must be labelled with its optimal arrival time, \( t(j) \) and optimal predecessor, \( p(j) \). The optimal arrival time labels give the minimum time at which each node can be reached and from this a minimum-time field function, giving the minimum time at which every point in the domain can be reached, can be approximated. Once minimum arrival time is known as a function of position, isochrones are defined by the contours of this function.

Isochrones are thus generated in two phases: approximation of the minimum-time field and calculation of contours of this field.

Let \( f(x) \) be the minimum time at which the position \( x \) can be reached. The problem of approximating this minimum-time function is solved by fitting a rectangular finite-element mesh to the data points provided by the optimal path algorithm. The field is defined by the values at the vertices of a rectangular grid and the field values at points other than these vertices are given by interpolation using basis functions in a similar manner to the interpolation used for wind and current, that is

\[
f(x) = \sum_{i=1}^{n} \psi_i(\xi(x))f_i(j(x)),
\]

(4.1)
where \( \psi_i(\xi) \) is one of the \( n \) basis functions, \( j(x) \) is the element containing \( x \), \( \xi(x) \) is the element coordinate vector corresponding to \( x \) for this element, and \( f_i^j \) is the value of the field at vertex \( i \) of element \( j \). For rectangular elements, \( j(x) \) and \( \xi(x) \) can be found by

\[
j(x) = \begin{bmatrix} \frac{x - x_{\min}}{\Delta x} \\ n_x \frac{y - y_{\min}}{\Delta y} \end{bmatrix}; \\
\xi(x) = \left( \frac{x - x_1^{j(x)}}{\Delta x}, \frac{y - y_1^{j(x)}}{\Delta y} \right)^T,
\]

where \( x_{\min} \) and \( y_{\min} \) are the coordinates of the lower, left-most corner of the grid, \( \Delta x \) and \( \Delta y \) are the distances between vertices in the \( x \) and \( y \) directions respectively, \( n_x \) is the number of elements in one row of the grid, and \( x_i^j \) and \( y_i^j \) are the coordinates of the \( i \)th vertex of element \( j \). A simple example, with four elements, is shown in Figure 4.1.

![Figure 4.1 Example of a rectangular, bilinear finite-element mesh.](image)
In this example the basis functions are the same bilinear basis functions used for the wind and current interpolation, so each element has four vertices. Note that \( n_x = n_y = 2 \), where \( n_x \) and \( n_y \) are the number of elements in the \( x \) and \( y \) directions, respectively.

For each node position \( x_d, d \in N(G) \), we know the minimum arrival time, so we want to find the function \( f(x) \) such that \( f(x_d) = t(d), \forall d \in N(G) \). There exist many such functions, but we restrict our attention to the class of functions representable using our basis functions, that is functions of the form (4.1). However, this is not achievable in general because the problem is over-determined. We thus desire a function of the form (4.1) that most closely matches our data points, and the measure used to determine this closeness of fit is the sum of the squares of the differences between the data points \( t(d) \) and function values \( f(x_d) \). Thus, we wish to find \( f(x) \) such that the error \( S \), where

\[
S = \sum_{d \in N(G)} w_d [f(x_d) - t(d)]^2,
\]

is minimised. The \( w_d \) that appears in this expression is the weight assigned to node \( d \). We will set \( w_d = 1, \forall d \in N(G) \) and ignore it in all further analysis.

In order to minimise \( S \), we consider each element independently and define the error of element \( j \) to be \( S_j \), where

\[
S_j = \sum_{d \in E_j} w_d [f(x_d) - t(d)]^2,
\]

and \( E_j \) is the set of nodes contained in element \( j \), that is \( E_j = \{d \in N(G): j(x_d) = j\} \).

Because we wish to minimise the error by choosing the values at the vertices, we set the derivative with respect to the values at the vertices of the element equal to zero, that is
4.1 Isochrone Generation

\[ 0 = \frac{\partial S_j}{\partial f_{i}^{j}} \]
\[ = \frac{\partial}{\partial f_{i}^{j}} \sum_{dE_j} [f(x_d) - t(d)]^2 \]
\[ = 2 \sum_{dE_j} [f(x_d) - t(d)] \frac{\partial}{\partial f_{i}^{j}} [f(x_d) - t(d)] \]
\[ = 2 \sum_{dE_j} [f(x_d) - t(d)] \frac{\partial f(x_d)}{\partial f_{i}^{j}} \]

Substituting the expression for \( f(x) \), (4.1), gives

\[ 0 = 2 \sum_{dE_j} \left[ \sum_{k=1}^{n} \psi_k (\xi(x_d)) f_k^{j} - t(d) \right] \frac{\partial}{\partial f_{i}^{j}} \sum_{k=1}^{n} \psi_k (\xi(x_d)) f_k^{j} \]
\[ = 2 \sum_{dE_j} \left[ \sum_{k=1}^{n} \psi_k (\xi(x_d)) f_k^{j} - t(d) \right] \psi_j (\xi(x_d)) \]
\[ = 2 \sum_{dE_j} \left[ \sum_{k=1}^{n} \psi_k (\xi(x_d)) f_k^{j} \psi_j (\xi(x_d)) - 2 \sum_{dE_j} t(d) \psi_j (\xi(x_d)) \right] \]
\[ = 2 \sum_{k=1}^{n} \left[ \sum_{dE_j} \psi_k (\xi(x_d)) \psi_j (\xi(x_d)) \right] f_k^{j} - 2 \sum_{dE_j} t(d) \psi_j (\xi(x_d)) \]

Rearranging gives

\[ \sum_{k=1}^{n} \left[ \sum_{dE_j} \psi_k (\xi(x_d)) \psi_j (\xi(x_d)) \right] f_k^{j} = \sum_{dE_j} t(d) \psi_j (\xi(x_d)) \]

or

\[ A_{ik}^{j} f_k^{j} = b_{i}^{j} , \quad (4.2) \]

where \( A_{ik}^{j} \) is the matrix of which the \( ik^{th} \) element is \( \sum_{dE_j} \psi_k (\xi(x_d)) \psi_j (\xi(x_d)) \) and \( b_{i}^{j} \) is the vector of which the \( i^{th} \) element is \( \sum_{dE_j} t(d) \psi_j (\xi(x_d)) \). This is a square system of \( n \) linear equations for the values at the vertices of element \( j \), where \( n = 4 \) for bilinear elements. There is one such system for each element in the mesh, but these systems cannot be solved independently because adjacent elements have vertices in common. In
order to solve for all vertices simultaneously, a single system of equations is assembled from the systems for each element.

If the set $E_j$ is empty, then $A_{ik}^j = b_i^j = 0$ and the system of equations (4.2) has no solution. In this case the element $j$ is excluded from the assembly. This means that the variables solved for in the assembled system of equations are the values at the vertices of elements for which $E_j \neq \emptyset$, that is \{f_i^j : E_j \neq \emptyset, i = 1, \ldots, n\}. Note that, because vertices can be members of more than one element, $E_j = \emptyset$ does not necessarily mean that the values of $f_i^j, i = 1, \ldots, n$ are not solved for in the assembled system of equations.

This assembled system is solved using the LAPACK (Anderson et al 1995) routine DGESV.

An outstanding issue is the determination of the parameters defining the mesh, that is $x_{\min}$, $\Delta x$, $n_x$, and $n_y$. It is assumed that these parameters are chosen to ensure that the mesh is only just large enough to cover the graph. Thus, $x_{\min} = \min_{d \in N(G)} x_d$, $y_{\min} = \min_{d \in N(G)} y_d$, $x_{\min} + n_x \Delta x = \max_{d \in N(G)} x_d$, and $y_{\min} + n_y \Delta y = \max_{d \in N(G)} y_d$. The latter two constraints provide a relationship between $n_x$ and $\Delta x$, and also $n_y$ and $\Delta y$, meaning that the choice of one dictates the choice of the other. Because $n_x$ and $n_y$ must be integers, these are used as parameters in preference to $\Delta x$ and $\Delta y$.

Increasing the number of elements, and thereby reducing the spacing between vertices, allows for more flexibility in the form of the fitted function $f(x)$. However, the smaller the elements become, the smaller the average number of nodes per element becomes, and the likelier it becomes that any given element will contain no nodes and be excluded from the assembled system of equations. In the extreme case, the number of elements for which $E_j \neq \emptyset$ is equal to the number of nodes. An example of this is shown in Figure 4.2. In this diagram, the black circles depict nodes, and elements containing at least one node are shaded. Note that the distribution of nodes in this example is intended to simulate a graph with five stages (including the first and last stages that have one node each).
A more appropriate mesh for this distribution of nodes is shown in Figure 4.3. It can be seen that in this case all elements contain at least one node.

**Figure 4.2** Example of a too-fine mesh.

**Figure 4.3** Example of a more appropriately spaced mesh.
An iterative procedure is used to determine the number of elements. An initial, coarse, mesh is refined (by increasing the number of elements) until just before the number of empty elements exceeds a specified maximum.

Once the values at the vertices, $f_i^j$, are known $\forall j,i$, the function $f(x)$ is defined and contours can be computed. This is achieved using the GCONTR routine, written by Snyder (1978). An example of a fitted time field function and the corresponding isochrones are shown in Figure 4.4. The weather forecast used in this example is the same as that used in the examples demonstrating the effect of angle resolution in the previous chapter, that is a forecast provided the Whitbread Round The World Race fleet in January 1998.

**Figure 4.4** Fitted time field (indicated by colour variations), isochrones (black lines, labelled with time from start point in hours), and optimal route (red line).

We can see that the southerly parts of the isochrones tend to be closer to the destination than the respective northerly parts. This is to be expected because the optimal route follows a southerly course, and it is reasonable to expect the optimal route to be one that
tends to be closer to the destination than other routes at a given time. This is not guaranteed in general, however.

4.2 Summary

In this chapter, we have presented a method for generating surfaces of equal time, or isochrones, using information generated by the deterministic routing algorithm described in the previous chapter. A function defined by the interpolation of values on the vertices of a rectangular grid by bilinear basis functions is fitted to the arrival time data using least squares. Isochrones are then found by generating contours of this function.

The next chapter resumes the main discussion of weather routing. The concept of uncertainty in the weather is introduced and an algorithm for generating optimal routes that somewhat mitigate the effects of this uncertainty is described.
5. Stochastic Routing

The preceding analysis leading to the deterministic solution method makes use of the assumption that the weather conditions are known at every position and time of interest. This information is obtained from a weather forecast. However, it is very difficult to predict the weather with any accuracy, so it is likely that the weather at some time in the future will differ from the predicted weather. This chapter is concerned with developing a technique for finding the optimal strategy when sailing through uncertain weather.

Previous work has dealt with this by treating weather conditions as random variables (Zoppoli, 1972) or by scenario aggregation (Carryer, 1995). We use a branching scenario model of the uncertainty in the weather, so the approach of Carryer is applicable. However, the progressive hedging algorithm, as used by Carryer for scenario aggregation, is not guaranteed to yield the global optimum. It is our intent to present an alternative method for dealing with uncertain weather conditions.

5.1 Stochastic Model

If we are to accommodate uncertainty in the weather, this uncertainty must be quantified, and for this we need a model that describes how this uncertainty behaves. The simplest stochastic model treats the wind speed and direction at a point as a random variable with a known distribution. However, this model incorporates no information about what has happened at the past when, in fact, weather is serially correlated. For example, consider the weather at a point on the ocean with a region with strong wind to the west. This situation is illustrated in Figure 5.1.
In a simplified situation, there are three possibilities for the movement of the region of strong wind: moving directly east and therefore passing over the yacht, passing to the north of the yacht, or passing to the south of the yacht. Once it is known which of these three possibilities is occurring, it is known whether to expect strong or light wind. In other words, the weather at a given time depends on what has happened to the weather in the past.

Our stochastic model must therefore be able to model this serial correlation. The model we have chosen does this using a branching tree structure. A branch in this tree structure represents the arrival of new information and a node the actual information. The representation of the above example using this structure is shown in Figure 5.2. The node at time $t_1$ represents the situation as shown in Figure 5.1. The three nodes at time $t_2$ represent the three possibilities for the wind: light (strong to the north of the yacht), strong, or light (strong to the south of the yacht). Thus, in this example, there are three possible ways for the weather to change over time, corresponding to three possible paths through the tree. Each of these possible paths is called a scenario.

At time $t_1$ all three scenarios are indistinguishable, meaning that it is not possible to tell which will eventuate. The solid lines in this diagram represent the flow of information. A single line leaves the node at time $t_1$, indicating that the only information we have at that time is that one of those three scenarios will eventuate. The colouring of this line

**Figure 5.1 Example weather conditions before information about the movement of the region of strong wind is available.**
indicates this also. At time $t_2$, the solid line branches into three separate scenarios. This represents the availability of information at time $t_2$ about which of the three scenarios will eventuate. From time $t_1$ until time $t_2$ all three scenarios are said to be members of the bundle $\{1,2,3\}$.

Figure 5.2 Example scenario structure with a single branch and three scenarios. The colours red, blue, and green represent scenarios 1, 2, and 3, respectively.

The dashed lines connecting the information nodes represent temporal weather interpolation. This indicates that between the times of $t_1$ and $t_2$ there are three different ways of interpolating the weather. Recall, from §2.3, that the wind and current at a given time and position are obtained by interpolating known values that exist at discrete points in space and time. Therefore, to obtain weather information for a time $t_1 < t < t_2$, it is necessary to interpolate in time between the known values at time $t_1$ and those at time $t_2$. However, there are three known wind fields at time $t_2$, corresponding to the three scenarios. The consequence of this is that every calculation that requires information about the weather for times between $t_1$ and $t_2$ has three possible values, each with a known probability.

A weather forecast predicts how the weather in a certain area will change over time. When referring to weather, we will be primarily concerned with wind, although the
same arguments can be applied to current. In the presence of uncertainty, a forecast, or several forecasts, will predict a number of different possibilities for how the wind will change over time. These different possibilities correspond to scenarios in our stochastic model and each scenario \( s \) is assigned a probability \( p_s \). We will call the set of possible scenarios \( S \) and note that

\[
S = \{s_1, s_2, \ldots, s_{|S|}\},
\]

\[
\sum_{s \in S} p_s = 1,
\]

\[
0 \leq p_s \leq 1, \quad \forall s \in S
\]

A slightly more complex example of a branching scenario tree is shown in Figure 5.3. In this example new information is received at times \( t_2 \) and \( t_3 \).

![Figure 5.3](image)

**Figure 5.3** A branching scenario tree with three branches and four scenarios.

At time \( t_1 \) there is only one possibility for the wind, that is all four scenarios are indistinguishable. At time \( t_2 \) this bundle of four scenarios branches into two bundles of two scenarios each and at time \( t_3 \) all four scenarios are distinguishable. We will define
\( B(t) \) to be the set of bundles at time \( t \). In the example of Figure 5.3, \( B(t) = \{[1,2,3,4]\} \) for \( t_1 \leq t < t_2 \), \( B(t) = \{[1,2],[3,4]\} \) for \( t_2 \leq t < t_3 \), and \( B(t) = \{[1],[2],[3],[4]\} \) for \( t \geq t_3 \).

It is useful to consider the bundle that a scenario \( s \) belongs to at a certain time, \( t \). This is the bundle that is a member of \( B(t) \) and contains \( s \), and is defined by

\[
B(s,t) = B \in B(t) : s \in B. 
\]

In the example of Figure 5.3, \( B\{1,t\} \) for \( t_1 < t \leq t_2 \) is \( \{1,2\} \).

Wind, and maybe current, are now functions of scenario as well as position and time, that is \( w = w(x,t,s) \) and \( c = c(x,t,s) \), where \( s \) is the scenario. Note that for all scenarios in a bundle \( B \) the wind is identical, that is

\[
w(x,t,s_1) = w(x,t,s_2), \forall s_1, s_2 \in B, \forall B \in B(t). 
\]

Although it is possible to treat current as varying with scenario in this way, to do so is not necessary because current can be predicted with much greater accuracy than wind. Indeed, it is theoretically possible to have scenarios that differ only in current, but this is not as useful for weather routing as scenarios that differ in wind.

This dependence on scenario means that speed and therefore travel time are also functions of scenario, so (2.12) becomes

\[
s(\theta, x, t, s) = \| v(\phi^*(\theta, h(\theta, c(x,t,s))), w(x,t,s), c(x,t,s)) \| \\
= \| p(\phi^*(\theta, h(\theta, c(x,t,s))), w(\theta, t,s)) + c(x,t,s) \|_2,
\]

where \( h(\theta, c) = \| c \| \sin(\arg c - \theta) \).

The expressions for travel time of a segment (2.3) and of an arc (3.1) are modified similarly:

\[
c_{\text{seg}}(x_1, x_2, t, s) = \frac{d_{GC}(x_1, x_2)}{s(\theta_{GC}(x_1, x_2), x_1, t, s)},
\]

\[
c_{\text{arc}}(i, j, t, s) = c_{\text{seg}}(x_i, x_j, t, s)
\]
5.1 Stochastic Model

5.1.1 Climatology

In the times and areas of the ocean not covered by the weather forecasts, the scenario model is difficult to apply. We model the wind here by a random variable with a known distribution. This information is called *climatology* and gives the distribution of the wind as a function of position, \( W(x) \).

In the case of climatological weather information, the problem is much simplified because there is no serial correlation. That is, the weather distribution is not dependent on what has happened previously. In this case it is not necessary to maintain state information relating to scenario and a small modification of the deterministic method is sufficient. The backward-looking recursion, (3.4), becomes

\[
\begin{align*}
t(j) &= \min_{i \in \Gamma_j} \left[ t(i) + E_{w \in W(x_j)} \left( \hat{c}_\text{arc}(i, j, w) \right) \right], & j = n_{\text{start}} \\
p(j) &= \arg \min_{i \in \Gamma_j} \left[ t(i) + E_{w \in W(x_j)} \left( \hat{c}_\text{arc}(i, j, w) \right) \right], & j \neq n_{\text{start}}
\end{align*}
\]

where \( \hat{c}_{\text{arc}}(i, j, w) \) is the time taken to sail from node \( i \) to node \( j \) under the weather given by \( w \), and \( E_{w \in W(x_j)} \) is the operation of expectation over the random variable \( W(x_j) \).

The expectation operation is greatly simplified if \( W(x) \) is discretised, that is can only take on a finite number of values. If we define \( \hat{W}(x) \) to be the set of possible values of the random variable \( W(x) \), then the expectation becomes

\[
E_{w \in \hat{W}(x_j)} \left( \hat{c}_{\text{arc}}(i, j, w) \right) = \sum_{w \in \hat{W}(x_j)} P(\hat{W}(x_j) = w) \hat{c}_{\text{arc}}(i, j, w),
\]

where \( P(\hat{W}(x) = w) \) is the probability of the wind at position \( x \) being \( w \).

5.1.2 Problem Definition

The stochastic problem is to find the set of paths \( P^* = \{ P^*[s] : s \in S \} \), one for each scenario \( s \in S \), that minimises some cost function to be defined later. We will refer to
the path from this set corresponding to scenario \( s \) as \( P^*[s] \). While the scenarios in a
bundle, \( s \in B \), are indistinguishable, that is \( B \in B(t) \), where \( t \) is the time we are
considering, it is required that the paths for these scenarios are identical. The paths are
only permitted to differ when the scenarios are distinguishable.

We define the set of nodes in a path \( P \) that are reached by time \( t \) under scenario \( s \) to be \( P_t(P,s) \) and require that, for all nodes on paths corresponding to indistinguishable
scenarios at time \( t \), the nodes that are reached by time \( t \) are the same, that is

\[
P_t(P^*[s_1],s_1) = P_t(P^*[s_2],s_2), \forall s_1,s_2 \in B, \forall B \in B(t). \tag{5.1}
\]

The reason for this requirement is that at time \( t \) it is known which bundle in \( B(t) \) is
currently active, but it is not known which scenario within this bundle will eventuate.
Until it is known which scenario (or group of scenarios) will eventuate, the paths must
be identical. It is the goal in stochastic routing to find a path that hedges in some way
against this uncertainty.

In this section we consider the problem of finding the set of paths that takes the
minimum expected time. The expected cost (where cost is the time taken) of a set of
paths \( P \) is given by

\[
c_P(P,t_0) = \sum_{s \in S} p_s c_P(P[s],t_0,s),
\]

where \( c_P(P[s],t_0,s) \) is the cost of the path corresponding to scenario \( s \), \( P[s] \), if
scenario \( s \) eventuates.

Thus (3.2) becomes

\[
P^* = \arg \min_{P \in P_{\text{feas}}} c_P(P,t_{\text{start}}).
\]

Minimising the expected cost in this way leads to a solution that is called risk neutral
because no account is taken of the risk of the solution. For example, a minimum
expected time solution might perform very well under one likely scenario, but very
poorly under other, less likely scenarios. This is a risky solution because if one of the
less likely scenarios eventuates then a large cost will be incurred. A less risky solution
might perform worse under the more likely scenario, but not as badly under the less likely ones.

5.2 Solution Method

One path for each scenario is required, and in the absence of the path-matching constraint (5.1) it would be sufficient to calculate an optimal path for each scenario independently, using the technique described in §3.2. However, each of these solutions would require perfect foresight to anticipate the weather and so are likely to perform poorly in practice when the weather evolves differently from our predictions.

Recall the forward-looking recursion for the deterministic problem, (3.3):

\[
f^*(i,t) = \begin{cases} 
0, & i = n_{\text{finish}} \\
\min_{j \in \Gamma_i} \left[ c_{\text{arc}}(i, j, t) + f^*(j, t + c_{\text{arc}}(i, j, t)) \right], & \text{otherwise} 
\end{cases}
\]

\[
j^*(i,t) = \arg\min_{j \in \Gamma_i} \left[ c_{\text{arc}}(i, j, t) + f^*(j, t + c_{\text{arc}}(i, j, t)) \right], \quad i \neq n_{\text{finish}}
\]  

(5.2)

Here \( c_{\text{arc}}(i, j, t) + f^*(j, t + c_{\text{arc}}(i, j, t)) \) is the optimal time to go from node \( i \) at time \( t \) if the next node on the path is node \( j \). In the stochastic problem, this optimal time to go is uncertain; it depends on which scenario eventuates.

In order to accommodate this uncertainty, it is necessary to know, in addition to the position and time, where in the scenario tree we are. This requires the addition of another state variable that identifies the branches that have been taken. It is convenient to identify the branch history by specifying a scenario. For example, consider the scenario tree shown in Figure 5.3. At times between \( t_1 \) and \( t_2 \) there is only one possible state for the branch history, that is the bundle \( \{1,2,3,4\} \); at these times any scenario identifies the same state. At times between \( t_2 \) and \( t_3 \), we have received information about which of the bundles \( \{1,2\} \) or \( \{3,4\} \) will eventuate. In this case, a scenario-state of 1 or 2 means that we know that the bundle \( \{1,2\} \) will eventuate and of 3 or 4 means that we know \( \{3,4\} \) will. For the stochastic case, we will extend the definition of a state to include a scenario component as well as position and time, and refer to node-time-scenario triples of the form \( (i,t,s) \), where \( i \) is the node, \( t \) is the time, and \( s \) is the
scenario indicating which bundle we know to be eventuating. Thus, the bundle we know
to be eventuating when in state \((i,t,s)\) is \(B(s,t)\).

From the node-time-scenario triple \((i,t,s)\) we know that the only scenarios that can
possibly eventuate are those in the same bundle as \(s\) at time \(t\), that is \(s' \in B(s,t)\), but
we don’t know which. Thus, we want to choose the successor to node \(i\), \(j^*(i,t,s)\), to
give the least expected time to go, where the (conditional) expectation is taken over the
scenarios \(s' \in B(s,t)\). We will define \(t^*(i,j,t,s)\) to be the expected optimal time to go
from node \(i\), at time \(t\), in scenario \(s\), when passing through node \(j\) next, which can be
written

\[
t^*(i,j,t,s) = \sum_{s' \in B(s,t)} \frac{p_{s'}}{p_{B(s,t)}} \left[ c_{arc}(i,j,t,s') + f^*(j,t + c_{arc}(i,j,t,s'),s') \right],
\]

where \(p_{B(s,t)} = \sum_{s' \in B(s,t)} p_{s'}\) is the probability of the bundle containing scenario \(s\) at time \(t\)
and \(p_{s'}/p_{B(s,t)}\) is the conditional probability of scenario \(s'\) at time \(t\) given that it is
known that one of the scenarios in the bundle containing scenario \(s\) will eventuate.

Because all of the scenarios \(s' \in B(s,t)\) are indistinguishable at time \(t\), one might
expect the travel time \(c_{arc}(i,j,t,s')\) to be the same for all \(s'\). The reason this term is
included in the expectation is that, due to the temporal weather interpolation, the travel
time depends on which scenario in the bundle eventuates. It will be shown that this has
no effect on our ability to satisfy the path matching constraint (5.1).

Modifying (5.2) to choose the successor to node \(i\) that minimises the expected optimal
time to go gives the stochastic forward-looking recursion:

\[
\begin{align*}
f^*(i,t,s) &= \begin{cases} 
0, & i = n_{\text{finish}} \\
\min_{j \in \Gamma_i} t^*(i,j,t,s), & \text{otherwise}
\end{cases}, \\
j^*(i,t,s) &= \arg\min_{j \in \Gamma_i} t^*(i,j,t,s), \quad i \neq n_{\text{finish}}
\end{align*}
\]

(5.4)

It can be seen that (5.4) gives the same optimal successor, \(j^*\), and optimal expected
time to go, \(f^*\), for all scenarios in the same bundle at time \(t\), that is
5.2 Solution Method

$j^*(i,t,s_1) = j^*(i,t,s_2), \forall s_1, s_2 \in B, \forall B \in B(t)$. This follows from the fact that $B(s_1,t) = B(s_2,t), \forall s_1, s_2 \in B(s,t)$. Thus, the path matching constraint (5.1) is satisfied.

5.2.1 Implementation

In the same manner as for the deterministic problem, it is possible to define an order in which each node can be labelled using (5.4). Recall that such an order is that of decreasing rank, but that an infinite number of calculations are required because the optimal time to go must be evaluated for all possible times at each node. This problem remains for the stochastic problem; the only difference is that the optimal time to go must also be evaluated for every scenario bundle at each node.

This problem was avoided for the deterministic problem by switching to a backward-looking recursive definition, (3.4). This technique is not suitable for the stochastic problem for a number of reasons.

Because we are minimising expected time to reach the destination in the stochastic case, some compromise is involved because it is unlikely we will find a path that takes the minimum time under all scenarios. It follows that the optimal path may pass through a node at a time other than the earliest possible time at which that node can be reached. Moreover, the optimal path will pass through each node at different times, in general, depending on which scenario eventuates. For this reason, it is not possible to consider only one state at each node, as was the case in the deterministic problem.

In addition, the backward-looking technique is not readily adapted to enforce the path matching constraint, (5.1), because the choice of a predecessor has no direct control over the branching of the path. In order to have this control, it is necessary to control the choice of successors, as in (5.4), the forward-looking recursion.

Finally, at any node-time-scenario triple we must be able to calculate the expected time to go under an optimal policy. When finding optimal predecessors, as in the deterministic backward-looking recursion, the objective function becomes arrival time and thus the time to go from a node is not known. Unlike the deterministic problem, arrival time and time to go are not interchangeable objectives for the stochastic problem; minimising the time at which we arrive at a node does not necessarily minimise the expected time to reach the destination.
5.2 Solution Method

Recall that this leaves us with the problem of having an infinite number of states for which to evaluate times to go. Clearly, not every node-time-scenario triple is of interest when finding the path with minimum expected time to go. Recall from the deterministic routing section that we assume that the yacht will always be sailed as fast as possible and that there is no waiting at nodes, that is the departure time at a node is equal to the arrival time. This assumption means that there are a finite number of node-time-scenario triples that can be reached. All such node-time-scenario triples for an example are shown in Figure 5.4. The scenario structure used is that of the example shown in Figure 5.3.

![Figure 5.4](image)

**Figure 5.4** All possible arcs and node-time-scenario triples for an example with four scenarios.

This diagram follows the same form as for the deterministic case, the major difference being the branching of the time axis on each node. This branching corresponds to the branching in the scenario tree as illustrated in Figure 5.5. Note that this axis on each node is effectively a combined time and scenario axis. The tick marks represent discrete states.
Thus, it is possible to implement a forward-looking algorithm that takes finite time by labelling only those node-time-scenario triples that are reachable. An obvious method for doing this is to evaluate (5.4) recursively. This, trivially, gives the following algorithm:
Algorithm 5 Calculate $f^* = f^*(i,t,s)$ and $j^* = j^*(i,t,s)$

1. If $i = n_{\text{finish}}$ then
   a) $f^* \leftarrow 0$
   b) Stop
2. $f^* \leftarrow \infty$
3. For each node $j \in \Gamma_i$
   a) Calculate $t^* = t^*(i,j,t,s)$
      i) $t^* \leftarrow 0$
      ii) For each $s' \in B(s,t)$
         (a) Calculate $c_{\text{arc}}(i,j,t,s')$
         (b) Calculate $f^* (j,t + c_{\text{arc}}(i,j,t,s'),s')$
         (c) $t^* \leftarrow t^* + \frac{p_j}{P_{B(s,t)}} \left[ c_{\text{arc}}(i,j,t,s') + f^* (j,t + c_{\text{arc}}(i,j,t,s'),s') \right]$
   b) If $t^* < f^*$ then
      i) $f^* \leftarrow t^*$
      ii) $j^* \leftarrow j$

We will now look more closely at the example shown in Figure 5.4 as it demonstrates a number of important features of this algorithm.

The simplest iteration of the algorithm occurs when no branching occurs between a node and the successor of interest. In our example, this occurs when considering the arc $(1,2)$. The portion of Figure 5.4 relevant to this arc is shown in Figure 5.6.
In this case the calculation of the time to go from node 1 at time \( t_1 \) under any scenario \( s \in B(t_1) \) passing through node 2, \( t^*(1,2,t_1,s) \), is simply the sum of the travel time from node 1 to node 2, \( c_{seg}(x_1, x_2, t_1, s) \), and the time to go from node 2, \( f^*(2, t_1 + c_{seg}(x_1, x_2, t_1, s), s) \). The time at which we arrive at node 2, \( t_2 \) (\( = t_1 + c_{seg}(x_1, x_2, t_1, s) \)), is earlier than the time at which the bundle \( \{1,2,3,4\} \) branches into the bundles \( \{1,2\} \) and \( \{3,4\} \), so there is no need to take an expectation when labelling node 1. Also, because node 1 is left at a time at which the weather is defined, no interpolation of the weather in time is necessary to calculate \( c_{seg}(x_1, x_2, t_1, s) \).

Next we consider the arc \( (2,5) \). This arc is shown in Figure 5.7 and demonstrates two important features of the algorithm. The most obvious is that the arc splits into two. This is because node 5 is reached at a time later than the time at which the bundle \( \{1,2,3,4\} \) branches into the bundles \( \{1,2\} \) and \( \{3,4\} \), so an expectation is required to obtain the expected time to go from node 2 when passing through node 5, \( t^*(2,5,t_2,s) \). The other feature is that node 5 is reached at different times under each of the two bundles. This is due to the fact that at time \( t_2 \) there are two possibilities for the interpolated weather, represented by the two dashed lines joining the bundle \( \{1,2,3,4\} \) and the bundles \( \{1,2\} \) and \( \{3,4\} \) in Figure 5.3.

Figure 5.6 The arc \( (1,2) \) from the example of Figure 5.4.
The only significantly time-consuming work done by this algorithm is the calculation of \( c_{arc}(i, j, t, s') \), the travel time, in step (3)i(iii)(a). By calculating the number of times this calculation is performed, we can quantify the time-complexity of the algorithm.

For a given node (not the finish node \( n_{\text{finish}} \)), the number of travel times that must be computed is equal to the number of times at which that node can be reached, multiplied by the number of successors that node has. The number of times at which a node can be reached is equal to the sum of the number of times each of that node’s predecessors can be reached.

Let \( t_i \) be the number of times at which node \( i \) can be reached and \( C_i \) be the number of travel times that must be calculated at node \( i \). Then \( C_i = t_i \sum_{j \in \Gamma_i} t_j \) and \( t_i = |S| \sum_{j \in \Gamma_i} t_j \), as worst-case estimates. If we assume that there are \( N \) nodes of each rank \( 2 \leq r < r(n_{\text{finish}}) \) (and one node each of ranks 1 and \( r(n_{\text{finish}}) \), that is \( n_{\text{start}} \) and \( n_{\text{finish}} \), respectively) then we have

\[
C_i = \begin{cases} 
  t_i |S| N, & 1 \leq r(i) < r(n_{\text{finish}}) \\
  0, & r(i) = r(n_{\text{finish}}) 
\end{cases}
\]

and \( t_i = |S|^{r(i)} N^{r(i)-1} \). Thus, letting \( M = r(n_{\text{finish}}) \),

\begin{enumerate}
\item \textbf{Figure 5.7 The arc (2,5) from the example of Figure 5.4.}
\end{enumerate}
\[ C_i = \begin{cases} |S|^{r(i)-1}N^{r(i)}, & 1 \leq r(i) < M \\ 0, & r(i) = M \end{cases} \]

The total number of calculations required, \( C \), is thus \( \sum_{i \in N_G} C_i \), that is

\[
C = \sum_{i \in N(G)} \sum_{r(i) = 1}^{M-1} |S|^{r(i)-1}N^{r(i)}
= \sum_{r=1}^{M-1} N^r \sum_{i \in R_r} |S|^{r+1}
= \sum_{r=1}^{M-1} N^r (|S|N)^{r+1}

The time-complexity of the algorithm is \( O(C) \), that is

\[
O(C) = O\left( \sum_{r=1}^{M-1} (|S|N)^{r+1} \right)
= O(|S|N^2) + O(|S|N^3) + \ldots + O(|S|N^{M-1}) + O(|S|N^M)
= O(|S|N^M)
\]

An algorithm can be regarded as being good if it is \( O(N^c) \), where \( c \) is a constant. This is known as a polynomial-time algorithm. We have shown that the stochastic algorithm, however, is exponential-time, meaning that the time taken increases exponentially with the problem size, because \( M \) is dependent on the size of the problem. It is desirable, therefore, to reduce the complexity of the algorithm.

Improvements in the performance of the algorithm can be achieved by limiting the time state at each node \( i \) and scenario \( s \) to members of the set \( T_{i,s} \), containing \( |T_{i,s}| = T \) discrete values, where \( T \) is constant for all nodes and scenarios. When the time to go at a time \( t \) is requested, the time to go at the corresponding time \( T_{i,s}[\tilde{t}], \tilde{t} = 1,2,\ldots,T \) is calculated. The mapping of continuous times to discrete time states is discussed after the complexity analysis below.
Once the time to go (and optimal successor) has been calculated for a time $T_{i,s}[\bar{r}] = 1,2,\ldots,T$, it is stored in $f_{i,s}[\bar{r}]$ (and $j_{i,s}[\bar{r}]$) and reused if required. The algorithm then becomes

**Algorithm 6** Calculate $f^* = f^*(i,t,s)$ and $j^* = j^*(i,t,s)$ using discrete time states

1. If $i = n_{\text{finish}}$ then
   a) $f^* \leftarrow 0$
   b) Stop
2. Find the time state $T_{i,s}[\bar{r}] = 1,2,\ldots,T$ corresponding to $t$
3. If $f_{i,s}[\bar{r}]$ and $j_{i,s}[\bar{r}]$ are known then
   a) $f^* \leftarrow f_{i,s}[\bar{r}]; \quad j^* \leftarrow j_{i,s}[\bar{r}]$
   b) Stop
4. $f^* \leftarrow \infty$
5. For each node $j \in \Gamma_i$
   a) Calculate $t^* = t^*(i,j,T_{i,s}[\bar{r}],s)$
      i) $t^* \leftarrow 0$
      ii) For each $s' \in B(s,T_{i,s}[\bar{r}])$
         a) Calculate $c_{\text{arc}}(i,j,T_{i,s}[\bar{r}],s',s)$
         b) Calculate $f^*(j,T_{i,s}[\bar{r}]) + c_{\text{arc}}(i,j,T_{i,s}[\bar{r}],s',s')$
         c) $t^* \leftarrow t^* + \frac{p_x}{p_{i,x}(T_{i,s}[\bar{r}])} [f^*(j,T_{i,s}[\bar{r}]) + c_{\text{arc}}(i,j,T_{i,s}[\bar{r}],s',s')]$
   b) If $t^* < f^*$ then
      i) $f^* \leftarrow t^*; \quad j^* \leftarrow j$
6. $f_{i,s}[\bar{r}] \leftarrow f^*; \quad j_{i,s}[\bar{r}] \leftarrow j^*$

This means that the maximum number of times at which a node can be reached under each scenario is $T$, for a total of $|S|^T$ states, so $C_i = |S|^T \times |S|N = |S|^2TN$, thus
5.2 Solution Method

\[ C = |S|N + |S|^2 \sum_{r=2}^{M-1} \sum_{n \in R_r} TN \]
\[ = |S|N + |S|^2 (M - 2)TN . \]
\[ = |S|N + |S|^2 (M - 2)TN^2 . \]

The time-complexity of the algorithm is now

\[
O(C) = O\left(|S|N + |S|^2 (M - 2)TN^2 \right)
\]
\[
= O\left(|S|N\right) + O\left(|S|^2 MTN^2 \right) - O\left(2|S|^2 TN^2 \right).
\]
\[ (5.5) \]

This shows that the time taken for the algorithm increases quadratically with number of scenarios and number of successors and linearly with the rank of \( n_{\text{finish}} \) and number of time states. As in the deterministic case, the number of stages and the number of nodes in each are linearly related for a given problem domain due to the angle resolution constraint (3.5), so this gives us an approximately cubic relation between problem size and solution time, that is \( O(C) = O\left(|T|S^2 M^3 \right) \).

Figure 5.8 shows the reuse of labels on node 6 in our example. Node 6 is reached under scenario 3 (green) at the same time state, \( t_5 \), from nodes 3 and 4, so this node-time-scenario triple \( (6, t_5, 3) \) need only be labelled once. The same applies to the node-time-scenario triple \( (6, t_5, 4) \).
One problem with this algorithm is that we do not know in advance what the members of the sets $T_{i,s}$ should be. The method used to overcome this difficulty generates these sets dynamically during the execution of the algorithm.

The number of elements of each set $T_{i,s}$, $T$, is constant, and can be considered a parameter of the algorithm. These sets are considered to consist of equally spaced times of the form $T_{i,s}[\bar{t}] = t_{i,s}^{\text{min}} + (\bar{t} - 1)\Delta t$, $\bar{t} = 1, 2, \ldots, T$, where $\Delta t$ is the constant time interval between time states and is also a parameter. Thus the members of the set $T_{i,s}$ are determined by the value of $t_{i,s}^{\text{min}}$, and it is this that is determined dynamically.

It is assumed that the time at which a node, $i$, is reached on the optimal path under a scenario $s$, although not necessarily the earliest time, is likely to be one of the earlier times at which it can be reached. Based on this assumption, the minimum time at each node and scenario, $t_{i,s}^{\text{min}}$, is set to be the earliest time at which that node can be reached under that scenario. Because this time is not known in advance, $t_{i,s}^{\text{min}}$ is adjusted as node $i$ is reached at different times under scenario $s$. This is incorporated in Algorithm 6, as part of step (2).
5.2 Solution Method

In order to avoid difficulties in determining scenario branching, the method for mapping times to time states was chosen to be a round down strategy. The index into the time state set is given by

\[ \tilde{t}(i, t, s) = \left\lfloor \frac{t - t_{i,s}^{\min}}{\Delta t} \right\rfloor + 1, \]

which means that, for a time to map to a valid index in the range \([l, T]\), it must be in the interval \([t_{i,s}^{\min}, t_{i,s}^{\min} + T\Delta t]\).

It is important to note that there is no longer any guarantee of finding the optimal solution when using this restricted state space, because it is possible for the optimal path to pass through node \(i\) under scenario \(s\) at a time \(t \geq t_{i,s}^{\min} + T\Delta t\). For this reason, the parameters \(T\) and \(\Delta t\) must be chosen so that both a sufficient time state resolution (controlled by \(\Delta t\)) and a sufficient time-state span (controlled by \(T\Delta t\)) are achieved.

If a node \(i\) is visited under scenario \(s\) at a time \(t \geq t_{i,s}^{\min} + T\Delta t\), it will not be labelled, because there is no corresponding time state. In this case, the optimal expected time to go is effectively infinite. Step (2) of Algorithm 6 can now be specified in more detail:
5.2 Solution Method

**Algorithm 7** Find the time index \( \bar{t} \in [1,T] \) and state \( T_{i,s}[\bar{t}] \) corresponding to time \( t \).

1. If \( t \geq t_{i,s}^{\text{min}} + T \Delta t \) then \( f^* \leftarrow \infty \); Stop
2. If \( t < t_{i,s}^{\text{min}} \) then
   a) \( \bar{t}' \leftarrow \left\lceil \frac{(t_{i,s}^{\text{min}} - t)}{\Delta t} \right\rceil + 1 \)
   b) If \( \bar{t}' \leq T \) then
      i) Move \( f_{i,s}^*[k] \rightarrow f_{i,s}^*[k + \bar{t}' - 1], k = 1, \ldots, T - \bar{t}' + 1 \)
      Move \( j_{i,s}^*[k] \rightarrow j_{i,s}^*[k + \bar{t}' - 1], k = 1, \ldots, T - \bar{t}' + 1 \)
   c) \( t_{i,s}^{\text{min}} \leftarrow t \)
3. \( \bar{t} \leftarrow \left\lceil \frac{(t - t_{i,s}^{\text{min}})}{\Delta t} \right\rceil + 1 \)
4. \( T_{i,s}[\bar{t}] = t_{i,s}^{\text{min}} + (\bar{t} - 1) \Delta t \)

A consequence of dynamically adjusting the set of available time states at each node and scenario in this manner is that it is possible for more than \( T \) time states to be labelled for each node and scenario. An example of this case is illustrated in Figure 5.9. The same node is shown being reached at four different times. The left-to-right order in the diagram indicates the order in which these times are encountered in the algorithm. In this example \( T = 3 \), but each of the four arrival times shown result in a time to go calculation. The bold section of the time axis on each visitation of the node represents the range of times for which a corresponding time state exists. The black circles represent labels, and the grey circles represent discarded labels.
In order to avoid this, and thus reduce the number of calculations required, it is desirable to label each node at early times first. In the example above, if the node was visited at time $t_1$ first, the visitation at time $t_5$ would not have required that the node be labelled at $t_5$.

5.3 Summary

In this chapter we have addressed the issue of uncertainty in the prediction of the weather by considering a stochastic description of the change of wind over time. This stochastic change was characterised by a branching scenario structure in order to model the serial correlation inherent in weather systems. The deterministic method described in the previous chapter was used as the basis of a forward-looking recursive algorithm for generating minimum expected time solutions.

In the next chapter we compare the solution produced by this method with deterministic and minimal distance solutions. In the following chapter we investigate approximations we can introduce in order to reduce the time required by the stochastic algorithm.
6. Comparison of Deterministic and Stochastic Solutions

In the previous chapter we developed a stochastic dynamic programming approach for routing in the presence of uncertainty, based on the deterministic solution method described in Chapter 3. In this chapter we compare the optimal solutions produced by the deterministic and stochastic solution methods under uncertain weather conditions.

Consider the contrived example of uncertain weather shown in Figure 6.1. Here the initial wind is a uniform 8 knot northerly. The weather then branches into two scenarios, one in which the wind is stronger towards the north and the other in which it is stronger toward the south. The probabilities of the two scenario are 0.75 and 0.25, respectively. In both cases the wind direction remains from the north.
Figure 6.1 Example of the evolution of the weather over time in two scenarios. Scenario 1 proceeds (a) to (b) to (d) with probability 0.75 and scenario 2 proceeds (a) to (c) to (e) with probability 0.25.

If we knew which of these two scenarios was going to eventuate, we could use a deterministic routing algorithm to obtain a minimal time route. However, such a route will perform badly if the other scenario eventuates. Figure 6.2 shows four routes for these weather conditions. The red and green routes are the deterministic minimal time
solutions for scenarios 1 and 2, respectively. The yellow route is the minimal distance route. The blue and purple routes are the stochastic solution. Note that they coincide at first and then diverge once it is known which scenario will eventuate.

![Diagram showing deterministic, minimal distance, and stochastic routes.](image)

**Figure 6.2** Deterministic (red and green), minimal distance (yellow), and stochastic (blue and purple) solutions.

The routes shown here are intuitive; the deterministic routes head towards the strongest wind, that is the route assuming scenario 1 takes a northerly course and that assuming scenario 2 takes a southerly course. The stochastic solution heads slightly north of the minimal distance route initially. When it becomes apparent which scenario is eventuating, the route is in a position to take advantage of either scenario. The position is better for scenario 1 because this scenario is more likely to eventuate than scenario 2. The expected travel time for each of these routes is shown in Table 6.1.
### Table 6.1 Optimal route times for the routes shown in Figure 6.2.

<table>
<thead>
<tr>
<th>Colour</th>
<th>Description</th>
<th>Optimal route time under scenario 1</th>
<th>Optimal route time under scenario 2</th>
<th>Expected optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Deterministic scenario 1</td>
<td>4d 04:59:13</td>
<td>4d 18:21:04</td>
<td>4d 08:19:41</td>
</tr>
<tr>
<td>Green</td>
<td>Deterministic scenario 2</td>
<td>4d 15:38:23</td>
<td>4d 03:17:20</td>
<td>4d 12:33:07</td>
</tr>
<tr>
<td>Yellow</td>
<td>Minimal distance</td>
<td>4d 09:20:36</td>
<td>4d 07:09:02</td>
<td>4d 08:47:42</td>
</tr>
<tr>
<td>Blue and Purple</td>
<td>Stochastic</td>
<td>4d 05:42:42</td>
<td>4d 07:06:27</td>
<td>4d 06:03:38</td>
</tr>
</tbody>
</table>

This table confirms that the deterministic routes perform badly when the assumed weather does not eventuate. The stochastic route performs nearly as well as the corresponding deterministic solution under each scenario, and slightly better than the minimal-time solution. The differences between all of the optimal route times are small for this problem because the optimal routes lie close to a straight line between the start and the destination. The differences would be more apparent over a longer course, however the differences exhibited here are sufficient for this demonstration.

The expected optimal route time shown in this table is simply the expectation of the optimal route times under each individual scenario. We can see that the stochastic solution has the best expected time, at just over two hours less than the next best, the deterministic solution assuming scenario 1. The expected time for the deterministic solution assuming scenario 2 is another four hours behind this. The reason the deterministic solution assuming scenario 1 does so much better than that assuming scenario 2 is that scenario 1 is more likely than scenario 2 and therefore has more influence on the expected time.
6.1 Summary
In this chapter we have compared the solutions produced by the deterministic and stochastic solution methods described in Chapters 3 and 5 respectively. The deterministic method produced the best solutions when the assumed weather conditions eventuated, but performed poorly when the other scenario eventuated. The stochastic method produced a solution that performed nearly as well as the corresponding deterministic solution under each scenario, and performed significantly better in the expectation.

The next chapter looks at the computation time needed to produce solutions using the stochastic solution method, and presents four techniques for reducing this time.
7. Computation Time Improvements

In Chapter 5, we developed an algorithm for the solution of the minimum expected time routing problem under uncertain weather. In the previous chapter we demonstrated the advantage this technique has over the deterministic method, described in Chapter 3, when the weather is uncertain. In this chapter, we investigate the time needed to compute solutions using this method. We consider four methods for reducing this time.

Table 7.1 shows the solution times for six different resolution graphs. The problem whose solution was timed is the same as that used in §3.3.2 to illustrate the effect of changing the graph resolution on the optimal path. The times are averages of five runs on an Intel® Pentium® II processor running at 300MHz with 128MB of RAM under Microsoft® Windows NT® 4.0. This computer is approximately twice as fast as an entry-level computer used for this purpose on a typical yacht at the time of writing.
Table 7.1 Solution times for different node and stage spacings.

<table>
<thead>
<tr>
<th>Stage spacing (NM)</th>
<th>Number of stages, $M$</th>
<th>Node spacing (NM)</th>
<th>Number of nodes per stage, $N$</th>
<th>Mean solution time (s)</th>
<th>Number of node-time-scenario triples labelled</th>
<th>Optimal route time (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10</td>
<td>30</td>
<td>21</td>
<td>3.0</td>
<td>331,044</td>
<td>4.976663</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>41</td>
<td>19.0</td>
<td>1,995,350</td>
<td>4.953716</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.5</td>
<td>81</td>
<td>120.8</td>
<td>12,528,377</td>
<td>4.92773</td>
</tr>
<tr>
<td>50</td>
<td>16</td>
<td>30</td>
<td>21</td>
<td>9.0</td>
<td>956,239</td>
<td>5.075562</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>41</td>
<td>60.4</td>
<td>6,244,455</td>
<td>4.996093</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.5</td>
<td>81</td>
<td>426.2</td>
<td>43,699,757</td>
<td>4.976174</td>
</tr>
<tr>
<td>25</td>
<td>31</td>
<td>30</td>
<td>21</td>
<td>27.8</td>
<td>2,827,187</td>
<td>5.374679</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>41</td>
<td>216.4</td>
<td>20,976,427</td>
<td>5.058663</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.5</td>
<td>81</td>
<td>1723</td>
<td>168,748,599</td>
<td>5.036393</td>
</tr>
</tbody>
</table>

These data suggest a linear relationship between the solution time and the number of node-time-scenario triples labelled. This is consistent with the time-complexity analyses of both the deterministic and stochastic algorithms. Solution time is plotted against number of node-time-scenario triples labelled in Figure 7.1. Logarithmic scales have been used for both axes in order to accommodate the large range in both solution time and number of node-time-scenario triples labelled. A least-squares linear fit is shown.
Figure 7.1 Linear fit between number of node-time-scenario triples labelled and solution time.

Because the number of node-time-scenario triples labelled is dependent only on the problem being solved, it is a more reliable measure of the time required to solve the problem than the actual elapsed time, which can be affected by other processes running on the computer. For this reason, all comparisons of the solution time of the algorithm that follow are performed using the number of node-time-scenario triples labelled and a solution time predicted using the fitted formula $y = 1.0180 \times 10^{-5} x$, where $y$ is the solution time and $x$ is the number of node-time-scenario triples labelled.

The weather forecast used in the generation of these solution times defines the wind on a grid rectangular in latitude and longitude. The spacing in this particular forecast is $1^\circ 15'$ of longitude and $51'$ of latitude. This corresponds to a spacing perpendicular to the orientation of the stages ranging approximately from 64.3NM to 56.6NM at the latitudes spanned by the graphs. This would suggest that the stage spacing of 25NM is the most appropriate of the three considered if we are to capture all information provided by the forecast about the variation of the wind in space.

In §3.3.2, the same example was considered. It was observed that, in order to obtain an acceptable solution with a stage spacing of 100NM, a node spacing of at most 30NM is required. This gives an angle resolution of $16.70^\circ$. However, this same angle resolution
gave a solution inconsistent with the other solutions when achieved with a stage spacing of 50NM and a node spacing of 15NM. In order to obtain consistent solutions for this problem, then, an angle resolution of somewhat less than 16.70° is required. For the case of a stage spacing of 25NM a node spacing less than 7.5NM is required to achieve this. The number of node-time-scenario triples labelled for a node spacing of 5NM is shown in Table 7.2. The time required to obtain the solution is predicted based on the linear relationship derived from the above data.

Table 7.2 Timing information for a graph of sufficiently fine resolution.

<table>
<thead>
<tr>
<th>Stage spacing (NM)</th>
<th>Number of stages, ( M )</th>
<th>Node spacing (NM)</th>
<th>Number of nodes per stage, ( N )</th>
<th>Labels</th>
<th>Predicted solution time (s)</th>
<th>Optimal route time (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>31</td>
<td>5</td>
<td>121</td>
<td>576,192,904</td>
<td>5716</td>
<td>5.026548</td>
</tr>
</tbody>
</table>

The solution time predicted for this solution is just over an hour and a half. It is likely that on hardware likely to be in use on a yacht (at the time of writing) that this would take at least three hours. Unless a dedicated computer is used, the elapsed time would be even longer than this because the computer would be in use for other purposes during the computation.

It is undesirable for the computation of the solution to take this long for two main reasons. Firstly, the optimal path tells the navigator what to do at the time at which the optimisation is started. Thus, when the solution is obtained after three hours the navigator is presented with what he or she should have done three hours ago. This can be mitigated somewhat by asking for the optimal path for the time at which the solution is expected to be available; however, the position of the yacht at this time is not known exactly.

The second, and perhaps more compelling reason, is that it is often inconvenient for the navigator to have to wait three hours before getting any routing information. The navigator may want to run the optimisation multiple times with different parameters, or might need to make a decision at short notice. Because the ultimate goal of this work is a commercial product, attractiveness to navigators is an important consideration.
7.1 Scanning Order

For these reasons we consider four methods for reducing the time needed to compute optimal solutions. The first deals with the order in which each node’s successors are labelled while the last three limit which of a node’s successors are considered by the algorithm.

7.1 Scanning Order

The order in which the possible successors of a node-scenario pair are examined in the minimisation of (5.4) can have a significant effect on the performance of the algorithm. This effect is a consequence of the technique used to limit the number of time states maintained at each node-scenario pair, as shown in Figure 5.9.

We have shown, in Chapter 5, that if a node-scenario pair is examined first at the earliest time it is ever examined, then the number of times to go calculated from that pair is a minimum because the minimum time state at each node-scenario pair, $t_{i,s}^{\min}$, is only set once.

The order in which node-time-scenario triples are visited is determined by the order in which the successors of each node are visited in step (5) of Algorithm 6. It is not possible, in general, to visit every node first at its earliest reachable time using this algorithm, because nodes are visited in a depth-first pattern. This pattern means that each node in a stage is completely labelled before any other unlabelled node in that stage is visited. Therefore, if a node $i$ is the first node of rank $r(i)$ to be visited, all nodes with rank greater than $r(i)$ are visited first at times obtained by passing through node $i$.

However, we can possibly reduce the number of times $t_{i,s}^{\min}$ is changed by visiting successors in increasing order of travel time. Although this strategy does not guarantee that a node-scenario pair will be visited first at the earliest time it will ever be visited, it means that node-scenario pairs tend to be visited at earlier times before later times.

A problem with this strategy is that the order of evaluation of successors depends on the weather at a node-time-scenario triple, and thus must be calculated at each of these triples that is visited. When the number of node-time-scenario triples labelled is of the order of $10^9$ this would introduce substantial overhead. We overcome this difficulty by
visiting successors in increasing order of distance from a node. This approximation has
the added advantage that the sorting is independent of the weather and can be performed
at the graph generation stage.

Table 7.3 shows the number of node-time-scenario triples labelled for various graph
resolutions when successors are sorted by distance compared to the number when
successors are left unsorted. Note that the numbers for the unsorted case are different to
those used above to demonstrate the linear relationship between this number and
solution time. This is because different time state discretisation parameters were used.

Table 7.3 Improvement the in number of node-time-scenario triples labelled when
successors are sorted by distance.

<table>
<thead>
<tr>
<th>Stage spacing (NM)</th>
<th>Node spacing (NM)</th>
<th>Labels (sorted by distance)</th>
<th>Labels (normal)</th>
<th>Improvement</th>
<th>Change in optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>30</td>
<td>281,993</td>
<td>597,474</td>
<td>52.80%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>2,358,184</td>
<td>5,702,192</td>
<td>58.64%</td>
<td>+0.00353%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>14,719,397</td>
<td>37,985,322</td>
<td>61.25%</td>
<td>+0.08334%</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
<td>1,085,674</td>
<td>2,743,009</td>
<td>60.42%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>8,600,689</td>
<td>21,664,771</td>
<td>60.30%</td>
<td>+0.41801%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>53,146,192</td>
<td>129,005,556</td>
<td>58.80%</td>
<td>-</td>
</tr>
<tr>
<td>25</td>
<td>30</td>
<td>3,036,101</td>
<td>9,323,796</td>
<td>67.44%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>28,418,636</td>
<td>68,676,891</td>
<td>58.62%</td>
<td>+0.35501%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>183,907,340</td>
<td>405,789,073</td>
<td>54.68%</td>
<td>+0.00014%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>544,828,426</td>
<td>1,158,277,325</td>
<td>52.96%</td>
<td>+0.18928%</td>
</tr>
</tbody>
</table>
It can be seen that sorting successors in this way consistently results in a reduction in the number of node-time-scenario triples labelled, and therefore in the solution time. The improvement ranges between 52.80% and 67.44% for the graph resolutions considered for this problem, although the 67.44% figure seems to be somewhat out of line with the other results. This discrepancy may be related to the fact that the graph that produced this result has the coarsest angle resolution of those considered, at 50.19°. Other than this, there is no apparent relationship between the improvement and graph resolution.

The optimal route times produced using this method are either the same as or slightly worse than those produced without sorting successors. The differences that are present can be attributed to the method used to map times onto discrete time states. In particular, (5.6) maps a range of times onto each time state. When a node-scenario pair is labelled at a time that maps onto a time state that has been previously labelled, the label associated with that time state is used instead of creating a new label. Therefore, it is possible for the optimal solution produced to depend on the order in which the nodes were labelled. The differences seen in Table 7.3 are not significant, being less than one percent. The greatest difference that we observe with this example is 0.42%, or approximately 30 minutes out of five days. When considering other methods for reducing the solution time, we will regard differences in the optimal route time of this magnitude as being attributable to this cause also.

### 7.2 Scanning Selection

Another technique we can use to reduce the amount of work required for Algorithm 6 is to reduce the number of successors of each node. The simplest way to do this is to reduce the number of nodes in each stage, but the angle resolution constraint (3.5) prohibits this approach. Three approaches that use a subset of the nodes in the next stage as successors are considered.

#### 7.2.1 Constant Angle Resolution

Recall that the argument leading to the angle resolution constraint uses the fact that the maximum angle spacing occurs at successors closest to the node under consideration, and it is this maximum spacing that is constrained. Because of this, it is possible for the angle spacing between more distant successors to be much less than this maximum
value. It was argued that the angle spacing must be sufficiently small because the decisions that characterise a path are those of the direction in which to head the boat. Conversely, a spacing much smaller than the minimum is of little value because directions separated by this spacing can not be distinguished, that is they are effectively the same decision. Therefore, once the maximum allowable angle spacing has been specified, it is only necessary to include sufficient successors such that this maximum is not exceeded.

Let $\beta_i^{jk} = \theta_{GC}(x_i, x_k) - \theta_{GC}(x_i, x_j)$ be the angle between the line joining node $i$ and node $j$ and that joining node $i$ and node $k$. The minimum angle spacing from node $i$ is thus $\beta_i^{\min} = \min_{k=1, \ldots, |R_i|} \beta_i^{R_i[k]R_i[k+1]}$. If the angle resolution constraint is satisfied, this minimum will be much less than the allowable maximum angle spacing.

An example of excluding nodes based on angle spacing is shown in Figure 7.2. In this example, node $i$ has rank 0, so its successor set is selected from nodes of rank 1. We assume that the spacing of the nodes in this stage $R_i$ has been chosen such that $\beta_i^{\min}$ is small enough to satisfy the angle resolution constraint. Then, in order for all nodes in the successor set, $\Gamma_i$, to satisfy the angle resolution constraint, that set need only include enough nodes so that the angle spacing never exceeds $\beta_i^{\min}$. The nodes $R_i[1]$ and $R_i[2]$ have the minimum angle spacing, so these nodes must be included. The dashed line labelled (2) represents an angle spacing of $\beta_i^{\min}$ from node $R_i[2]$. The node furthest from $R_i[2]$ for which the angle spacing does not exceed $\beta_i^{\min}$, that is that lies below the dashed line, is $R_i[3]$, so this node must be included in $\Gamma_i$. The dashed line labelled (3) shows an angle spacing of $\beta_i^{\min}$ from the node $R_i[3]$, the node we just included. In this case the nodes $R_i[4]$ and $R_i[5]$ can be excluded from the successor set because the angle spacing between nodes $R_i[3]$ and $R_i[6]$ is still less than $\beta_i^{\min}$, but $R_i[6]$ must be included because the angle spacing between nodes $R_i[3]$ and $R_i[7]$ is greater than $\beta_i^{\min}$. 
Figure 7.2 Example showing nodes excluded from a successor set on the basis of unnecessarily small angle spacing.

In this way, the number of successors considered from each node can be reduced such that the angle resolution constraint is just satisfied.

As for the successor sorting modification above, the number of node-time-scenario triples labelled when nodes are excluded to maintain a constant angle resolution for various graphs is compared to the unmodified number in Table 7.4.
Table 7.4 Improvement in the number of node-time-scenario triples labelled when successors are excluded so as to maintain a constant angle resolution.

<table>
<thead>
<tr>
<th>Stage spacing (NM)</th>
<th>Node spacing (NM)</th>
<th>Labels (constant angle resolution)</th>
<th>Labels (normal)</th>
<th>Improvement</th>
<th>Change in optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>30</td>
<td>106,099</td>
<td>597,474</td>
<td>82.24%</td>
<td>+0.04033%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>896,123</td>
<td>5,702,192</td>
<td>84.28%</td>
<td>+0.00353%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>5,920,385</td>
<td>37,985,322</td>
<td>84.41%</td>
<td>+0.04418%</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
<td>273,996</td>
<td>2,743,009</td>
<td>90.01%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1,898,676</td>
<td>21,664,771</td>
<td>91.24%</td>
<td>+0.41801%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>12,682,960</td>
<td>129,005,556</td>
<td>90.17%</td>
<td>-</td>
</tr>
<tr>
<td>25</td>
<td>30</td>
<td>419,704</td>
<td>9,323,796</td>
<td>95.50%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>3,983,669</td>
<td>68,676,891</td>
<td>94.20%</td>
<td>+0.35501%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>21,752,350</td>
<td>405,789,073</td>
<td>94.64%</td>
<td>+0.00014%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>65,192,379</td>
<td>1,158,277,325</td>
<td>94.37%</td>
<td>+0.18928%</td>
</tr>
</tbody>
</table>

The constant angle resolution modification provides very large improvements in the number of node-time-scenario triples labelled. The improvement becomes more pronounced as the stage spacing decreases, but appears to be unrelated to the node spacing.

The changes to the optimal route times are small enough to be attributed to differences in what time is first mapped to each time state, as discussed for the successor sorting case above. Thus, this method provides a very large speed-up without producing significantly worse solutions.
7.2.2 Distance
Consider the example shown in Figure 7.3, which shows a set of parallel stages. Node \( i \) is marked, as are all possible arcs to nodes in the next stage, that is all possible successors. The distance from node \( R_3[1] \) to node \( R_3[9] \) is four times the distance from node \( i \) to node \( R_3[9] \) (the diagram is to scale), so the length of the arc \((i,R_3[1])\) is \(2\sqrt{17} \approx 8.25\) times that of \((i,R_3[9])\). Note that, for simplicity, this calculation assumes that the nodes lie on a plane rather than the surface of a sphere.

\[ R_3[1] \]
\[ R_3[9] \]
\[ i \]

Figure 7.3 Example showing distance variation over a node’s successors with parallel stages.

Recall from §2.2 that the time taken to sail a segment between two points \( x_1 \) and \( x_2 \) at a time \( t \), \( c_{wc}(x_1,x_2,t) \), is calculated by assuming that the speed of the boat remains constant over the segment and that the great circle path is followed. The constant speed assumption is only valid for either short distances or constant weather conditions. In the case of distant successor nodes as in the example above, the assumption is only valid if the weather does not change significantly over that distance or the time taken to sail it. Thus it was decided to exclude nodes from successor sets upon generation of the graph, based on distance. Two methods for doing this are discussed.
7.2 Scanning Selection

The first method excludes a possible successor to a node if the distance between the node and the possible successor is greater than some maximum, \( d_{\text{max}} \). This maximum distance is specified in advance as a parameter to the graph generation algorithm. Mathematically:

\[
\Gamma_i = \{ j \in R_{i(i+1)} : d_{GC}(x_i, x_j) \leq d_{\text{max}} \}.
\]  

(7.1)

The second method excludes those possible successors to a node for which the distance between the node and the possible successor is greater than some scale factor, \( \sigma \), multiplied by the distance to the closest possible successor. Again, this scale factor is a parameter of the graph generation algorithm. This condition is equivalent to (7.1) where the maximum distance is calculated for each node \( i \) from the scale factor, \( \sigma \), and the minimum distance, that is

\[
d_{\text{max}} = \sigma \min_{j \in R_{i(i+1)}} d_{GC}(x_i, x_j).
\]

There are two benefits obtained by excluding distant successors. Reduction of inaccuracy due to violation of the constant speed assumption when calculating travel times has already been discussed. The other benefit is a reduction of the number of node-time-scenario triples that must be labelled in order for a node to be labelled. Reducing the number of successors from each node effectively decreases \( N \) in (5.5), so the time taken by the algorithm decreases approximately quadratically with the number of successors that can be excluded.

The number of node-time-scenario triples labelled when using the second exclusion criterion with a scale factor of \( \sigma = 2 \), that is excluding all successors more than twice as far away as the closest successor, is shown in Table 7.5 for a range of graph resolutions.
Table 7.5 Number of node-time-scenario triples labelled when excluding successors more than \( \sigma = 2 \) times as far away as the closest successor.

<table>
<thead>
<tr>
<th>Stage spacing (NM)</th>
<th>Node spacing (NM)</th>
<th>Labels (( \sigma = 2 ))</th>
<th>Labels (normal)</th>
<th>Improvement</th>
<th>Change in optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>30</td>
<td>114,930</td>
<td>597,474</td>
<td>80.76%</td>
<td>+0.04033%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1,149,710</td>
<td>5,702,192</td>
<td>79.84%</td>
<td>-0.00018%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>8,029,877</td>
<td>37,985,322</td>
<td>78.86%</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
<td>185,497</td>
<td>2,743,009</td>
<td>93.24%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>2,268,659</td>
<td>21,664,771</td>
<td>89.53%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>15,166,317</td>
<td>129,005,556</td>
<td>88.24%</td>
<td>-</td>
</tr>
<tr>
<td>25</td>
<td>30</td>
<td>270,379</td>
<td>9,323,796</td>
<td>97.10%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>2,175,017</td>
<td>68,676,891</td>
<td>96.83%</td>
<td>+0.00004%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>20,040,025</td>
<td>405,789,073</td>
<td>95.06%</td>
<td>+0.04503%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>63,264,727</td>
<td>1,158,277,325</td>
<td>94.54%</td>
<td>-0.10827%</td>
</tr>
</tbody>
</table>

Excluding distant nodes results in large improvements in the number of node-time-scenario triples labelled. The improvements become more pronounced as the stage spacing decreases because, for a given node spacing, distance increases more quickly at smaller stage spacings. This is illustrated for the case of a node spacing of 30NM in the graph of Figure 7.4.
This graph shows that as stage spacing increases for a given node spacing, the number of nodes less than twice as far away as the closest node, and therefore the number of nodes considered as successors, increases.

The changes in the optimal solution time are insignificant. It is interesting to note that for two of the graph resolutions considered the optimal solution time was improved by excluding distant nodes. The improvements, however, are insignificant and can be attributed to the same cause as that discussed for the successor sorting method above.

Thus, excluding distant successors provides a large improvement in the time taken to compute the optimal solution without significantly affecting the solution obtained.

### 7.2.3 Global Optimisation

Both the techniques discussed above for limiting the successor set of each node are intended to be applied when the graph is generated, before the solution algorithm is run. Thus, they do not utilise important information about the particular problem being solved, in particular the weather conditions and the boat’s velocity prediction function. When labelling a node, the expected optimal time to reach the destination is computed for each possible successor node and the minimum of these is chosen. This can be
regarded as a sub-problem – that of minimising the expected optimal time to go
function, $t^*(i, j, t, s)$, where $i$, $t$, and $s$ are known and fixed for each sub-problem. We
will refer to this function as $t_{im}^*(j)$ to emphasise that $j$ is the variable over which we
are minimising.

Because the set of successor nodes constitutes a discrete domain for this minimisation,
it is possible to find a solution by examining every successor node. Indeed, this is the
method used in Algorithm 6 and the ensuing analysis above. However, it may be
possible, given sufficient information about the function $t_{im}^*(j)$, to exclude successor
nodes on the basis of information obtained previously. Such an approach can be
regarded as analogous to minimising a continuous function in preference to complete
enumeration, which is impractically time-consuming.

If the function $t_{im}^*(j)$ is convex, then it has a single minimum and any local
minimisation technique will therefore yield the global minimum also. However, we do
not know that this function is convex, so in order to find the global minimum a global
minimisation technique is required.

One such technique is known as a covering method. We will first consider this
technique in general as applied to a continuous function of one variable, $f(x)$, defined
on the domain $x \in [a, b]$. When finding a global minimum computationally, it is often
infeasible to determine the exact minimum, that is $x^*: f(x^*) \leq f(x), \forall x,$ and a relaxed
problem is considered instead: find

$$
  x^*: f(x^*) \leq f(x) + \varepsilon, \forall x \in [a, b],
$$

(7.2)

where $\varepsilon > 0$ is a tolerance specified as a parameter to the algorithm and $x^*$ is a global
minimiser to within $\varepsilon$. This technique makes use of a lower envelope, $F(x)$, which is
known to satisfy

$$
  F(x) \leq f(x), \forall x \in [a, b].
$$

(7.3)

The lower envelope is updated during each iteration, as new values of $f(x)$ become
available, based on some knowledge about the behaviour of $f(x)$. The lower envelope
at iteration $k$ is written $F_k(x)$. The general covering method algorithm for a function of one dimension is:

**Algorithm 8** Covering method for global minimisation of $f(x)$ on the domain $x \in [a, b]$ with tolerance $\epsilon$ and starting point $x_1 \in [a, b]$.

1. $k \leftarrow 1$
2. Form lower envelope $F_k$ from $x_1, \ldots, x_k$
3. $x_{k+1} \leftarrow \arg \min_{x \in [a, b]} F_k(x)$
4. If $\min_{i=1,\ldots,k} f(x_i) - F_k(x_{k+1}) \leq \epsilon$ then Stop with $x^* = x_k$
5. $k \leftarrow k + 1$; Go to (2)

The stopping condition in this algorithm, step (4), depends on the parameter $\epsilon$. When $\min_{i=1,\ldots,k} f(x_i) - F_k(x_{k+1}) \leq \epsilon$, we have $\min_{i=1,\ldots,k} f(x_i) - \epsilon \leq F_k(x_{k+1})$. We also known that $F_k(x_{k+1}) \leq F_k(x), \forall x \in [a, b]$, so (7.3) becomes

$$f(x) \geq F_k(x), \forall x \in [a, b]$$

$$\geq F_k(x_{k+1}), \forall x \in [a, b]$$

$$\geq \min_{i=1,\ldots,k} f(x_i) - \epsilon, \forall x \in [a, b]$$

Thus, $\min_{i=1,\ldots,k} f(x_i)$ is a global minimum of $f(x)$ to within $\epsilon$, and $\arg \min_{i=1,\ldots,k} f(x_i)$ is the corresponding global minimiser.

We consider a covering method algorithm known as the Pijavskii-Shubert algorithm (Pijavskii, 1972 and Shubert, 1972). The function $f(x)$ is assumed to be Lipschitz-continuous on the domain $x \in [a, b]$, that is

$$|f(x_1) - f(x_2)| \leq L| x_1 - x_2 |, \forall x_1, x_2 \in [a, b],$$

(7.4)

where $L$ is the Lipschitz constant. This leads to a lower envelope at iteration $k$ of

$$F_k(x) = \min_{i=1,\ldots,k} \left[ f(x_i) + L| x_i - x | \right],$$

(7.5)
An example of this algorithm is illustrated for four iterations in Figure 7.5 through Figure 7.8. The function $f(x)$ has two local minima on the domain $[a,b] = [0,10]$. The initial point is chosen to be $x_1 = a + \frac{1}{2}(b - a) = 5$, and the initial lower envelope, $F_1(x)$, is shown in Figure 7.5. Note that the line segments comprising the lower envelope have slope $\pm L$. The lower envelope has two minima in this case; the minimum at $x = 0$ is chosen arbitrarily. The stopping condition test, $\min_{i=1}^{n} f(x_i) - F_1(x)$, is shown, and it is assumed that the tolerance $\varepsilon$ is smaller than this.

The next iteration is shown in Figure 7.6. The updated lower envelope, $F_2(x)$, is obtained by taking the maximum of the Lipschitz slope lines from $x_1$ and $x_2$ as in (7.5). In this case there is a unique minimum of $F_2(x)$ at $x = 10$. Again, we assume that the stopping condition is not satisfied.

Figure 7.5 Iteration $k = 1$

Figure 7.6 Iteration $k = 2$
Iterations 3 and 4 are shown in Figure 7.7 and Figure 7.8, respectively. The algorithm continues similarly until the stopping condition is satisfied.

In the case of the problem we wish to solve, that is minimise $t^*_m(j)$, the domain over which we are minimising is composed of a finite number of discrete values; specifically, $j \in \Gamma_i$. In order to apply the Pijavskii-Shubert algorithm to this problem we introduce a new function, $f(x)$, defined on a continuous domain $D$, where $D = \left[1, l|\Gamma_i|\right]$. The function $f(x)$ is defined such that $f(x) = t^*_m(\Gamma_i[x])$ for integral values of $x$. This allows us to use the Lipschitz continuity assumption. However, we constrain the minimisation of the lower envelope in step (3) of Algorithm 8 to consider only integral values of $x_{k+1}$, corresponding to values of $j$, that is

$$x_{k+1} \leftarrow \arg\min_{i=1, \ldots, |\Gamma|} F_k(x),$$

where $F_k(x)$ is defined as in (7.5).

In order to evaluate the lower envelope $F_k(x)$, the Lipschitz parameter, $L$, must be known. Given that $L$ is sufficiently large to satisfy (7.4), the algorithm will converge faster the smaller $L$ is. This suggests that the best value of $L$, which we will call $L^*$, is

$$L^* = \min\{L : |f(x_1) - f(x_2)| \leq L|x_1 - x_2|, \forall x_1, x_2 \in [a, b]\}.$$
For a general function $f(x)$, this value is difficult to determine, requiring evaluation of $f(x)$ at all values of $x \in [a, b]$. For this reason, we are not concerned with finding this optimal parameter, but simply with finding a small value of $L$ that satisfies (7.4). At the time of writing, the selection of the Lipschitz parameter is left to the user of the software.

Table 7.6 shows algorithm performance information when the Pijavskii-Shubert algorithm is applied with a very large Lipschitz parameter. The parameter was chosen so as to be large enough to ensure that all successors are considered. Thus, the improvements are due solely to changes in the order in which successors are labelled.

Table 7.6 Improvement in the number of node-time-scenario triples labelled when global optimisation is applied with a very large Lipschitz parameter.

<table>
<thead>
<tr>
<th>Stage spacing (NM)</th>
<th>Node spacing (NM)</th>
<th>Labels ($L = 10^6$)</th>
<th>Labels (normal)</th>
<th>Improvement</th>
<th>Change in optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>30</td>
<td>275,382</td>
<td>597,474</td>
<td>53.91%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>2,104,448</td>
<td>5,702,192</td>
<td>63.09%</td>
<td>+0.00353%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>12,310,160</td>
<td>37,985,322</td>
<td>67.59%</td>
<td>+0.08286%</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
<td>1,106,474</td>
<td>2,743,009</td>
<td>59.66%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>8,272,031</td>
<td>21,664,771</td>
<td>61.82%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>46,857,986</td>
<td>129,005,556</td>
<td>63.68%</td>
<td>-</td>
</tr>
<tr>
<td>25</td>
<td>30</td>
<td>3,240,486</td>
<td>9,323,796</td>
<td>65.24%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>28,565,308</td>
<td>68,676,891</td>
<td>58.41%</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>163,067,583</td>
<td>405,789,073</td>
<td>59.81%</td>
<td>+0.92209%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>439,101,416</td>
<td>1,158,277,325</td>
<td>62.09%</td>
<td>+1.00455%</td>
</tr>
</tbody>
</table>
The improvement in the number of node-time-scenario triples labelled achieved by this method is similar to that achieved by sorting successors by distance. This is to be expected because the Pijavskii-Shubert algorithm tends to explore areas containing local minima first.

The optimal route times are very close to the unsorted case. The times for a stage spacing of 25NM and node spacings of 7.5NM and 5NM differ from the unsorted case by more than we have seen when considering the other methods in this chapter, however. Because the differences are still very small, approximately one percent, we can safely regard them as insignificant.

Improvements from labelling nodes in a different order are side effects of the Pijavskii-Shubert algorithm. This approach is intended to be used with a small value of $L$ so that the number of successors considered from each node is reduced. We thus seek small values of $L$ for each of the graph resolutions we are considering.

The number of node-time-scenario triples labelled for various values of $L$, using a stage spacing of 100NM and a node spacing of 30NM is shown in Table 7.7.

Table 7.7 Different values of the Lipschitz parameter for a stage spacing of 100NM and node spacing of 30NM.

<table>
<thead>
<tr>
<th>Lipschitz parameter $L$</th>
<th>Continuity violations</th>
<th>Labels</th>
<th>Improvement</th>
<th>Change in optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>276,969</td>
<td>53.64%</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>276,747</td>
<td>53.68%</td>
<td>–</td>
</tr>
<tr>
<td>3.5</td>
<td>0</td>
<td>273,659</td>
<td>54.20%</td>
<td>–</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>270,918</td>
<td>54.66%</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>2200</td>
<td>201,082</td>
<td>66.34%</td>
<td>–</td>
</tr>
</tbody>
</table>
This table also includes the number of times a violation of the Lipschitz continuity assumption was detected. If this number is greater than zero then the Lipschitz parameter is too small. For the problem considered here, the smallest acceptable value of $L$ lies between 3 and 3.5. Because there is very little variation in the number of node-time-scenario triples labelled when changing $L$ from 10 to 3, we can take $L = 3.5$ to be the best value. The results of similar reasoning for the other graph resolutions we are considering for our example are shown in Table 7.8.

**Table 7.8** Number of node-time-scenario triples labelled for small values of the Lipschitz parameter that result in no violations of the continuity assumption.

<table>
<thead>
<tr>
<th>Stage spacing (NM)</th>
<th>Node spacing (NM)</th>
<th>Lipschitz parameter $L$</th>
<th>Labels</th>
<th>Improvement</th>
<th>Change in optimal route time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>30</td>
<td>3.5</td>
<td>273,659</td>
<td>54.20%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>3</td>
<td>2,028,342</td>
<td>64.43%</td>
<td>+0.57547%</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>3</td>
<td>12,026,812</td>
<td>68.34%</td>
<td>+0.08286%</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
<td>30</td>
<td>1,102,848</td>
<td>59.79%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>40</td>
<td>8,284,834</td>
<td>61.76%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>60</td>
<td>46,942,074</td>
<td>63.61%</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>30</td>
<td>40</td>
<td>3,242,809</td>
<td>65.22%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>60</td>
<td>28,432,017</td>
<td>58.60%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>80</td>
<td>162,824,370</td>
<td>59.87%</td>
<td>+0.92209%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>70</td>
<td>439,294,863</td>
<td>62.07%</td>
<td>+0.84533%</td>
</tr>
</tbody>
</table>

It can be seen that, when a small Lipschitz parameter is chosen such that the Lipschitz continuity assumption is never violated, there is very little difference between the number of node-time-scenario triples labelled using this value and using a very large
value. This suggests that the assumption of Lipschitz continuity is perhaps not suitable for characterising the function $f(x)$. However, it is interesting to note that in Table 7.7 the values of $L$ that resulted in continuity violations did not result in a change in the optimal route time. It may therefore be possible to use values of $L$ that result in continuity violations to obtain improvements without compromising the solution quality.

### 7.3 Summary

In this chapter we have considered four techniques for reducing the time taken to compute the optimal solution. The first sorts the successors of each node by distance from that node and achieves speed-ups of approximately 60%. The second and third exclude successors based on angle spacing and distance respectively. Both achieve speed-ups of approximately 90%. The fourth uses an adaptation of a global optimisation algorithm known as the Pijavskii-Shubert algorithm to achieve speed-ups of approximately 60%.
8. Conclusions

We have developed a technique for computing minimal expected time routes under uncertain weather conditions. To accomplish this we have discretised both the area of ocean and the time over which we consider sailing and applied a stochastic dynamic programming algorithm. We have modelled the uncertainty in the weather by a branching scenario tree in order to capture the serial correlation that exists due to the way weather evolves over time.

This technique has been shown to perform better in expectation than a deterministic method based on a similar technique. Therefore it is potentially useful in its intended application of on-yacht use during a race.

In order for the stochastic routing algorithm to be practicable for such a use, solutions must be computed in a reasonable amount of time. It was found that this was not the case for the simple case of considering all possible successors from each node, and modifications were considered, two of which yield sufficient improvement to render the technique practicable for use during a yacht race. Both of these modifications exclude some of the possible successors of each node. The first excludes distant successors and the second excludes successors such that the headings to the remaining successors are more evenly spaced. These modifications were shown, for one example, to not significantly affect the solution produced by the algorithm.

With one of these modifications incorporated, the stochastic routing algorithm has the potential to be a useful tool for the navigators of yachts competing in long, offshore races. For the algorithm to yield a solution that performs well under a large range of possible weather conditions, it is necessary to have a comprehensive scenario tree describing a sufficient number of possible scenarios, and to have probabilities assigned appropriately. Ideally, this information would be provided as part of a weather forecast, but such forecasts are not readily available at the time of writing. In its present implementation the software provides rudimentary tools for a navigator to combine and manipulate forecasts to form a scenario tree. It is hoped that, if stochastic weather routing becomes popular among navigators, this type of scenario-based forecast will become more widespread.
8.1 Future Work

Clearly this work is neither perfect, nor complete. There are a number of specific areas that could be developed further.

The assumption that the speed of the yacht remains constant when calculating the travel time between two points greatly simplifies this calculation, but introduces error that can be significant if the points are too far apart. This could be overcome by making use of the weather conditions en route in some way. A possible approach for this is to use a numerical integration procedure. While this would increase the amount of computation required by the algorithm, it would allow larger stage spacings, so it is possible that the computation time could be reduced.

A related area for improvement is in the model used to predict the velocity of a yacht in given weather conditions. At present, the only factors taken into account by this model are the wind and the current. While these are the major factors affecting the velocity of a yacht, other factors such as waves and general sea conditions can have a significant effect. No great difficulty is foreseen in the incorporation of such factors; they can modify the wind or current vector as appropriate.

Another assumption that could warrant further investigation is that one departs a node at the same time at which one arrived there. There are situations where it might be advantageous to wait at a node before heading off, for example in the presence of a strong tide it might not be possible to make any progress until the tide turns. The stochastic solution method can be modified to allow this by replacing the expression for the expected optimal time to go, \( t^* (i, j, t, s) \), defined in (5.3), by

\[
\begin{align*}
  t^* (i, j, t, s) &= \sum_{j \in B(i, s)} \frac{P_j}{P_{B(i, s)}} \left[ \min_{t' \geq t} \left( t' + f^* (j, t + t') \right) \right].
\end{align*}
\]

Here, instead of departing at the arrival time, \( t \), we take a minimisation over all times later than this. This modification would greatly increase the complexity of the algorithm, and would probably only be useful for relatively few races, but might be worth investigating for use in those cases when very strong tides or currents are expected.
The stochastic solution method we have developed produces minimal expected time solutions. The use of the expected time as the objective means that the solutions are risk neutral, in other words no account is taken of how risky the solution is. However, a navigator might be very interested in how risky a solution is, and his or her attitude towards this risk might depend on the yacht’s position in the race. For example, if the yacht is last, a risky solution might be preferred if it gives them a chance of overtaking the next boat. Such an attitude is termed risk seeking. Conversely, if the yacht is winning the race, a solution that plays it safe under all possible scenarios might be preferred, because it is less important to take the minimum amount of time than it is to remain ahead of the rest of the yachts. This attitude is a risk averse one.

Therefore, it would be useful for the user of stochastic routing software to be able to specify his or her attitude to risk in some way, and to have the algorithm take this into account. It is possible to do this in a crude way in the existing software by changing the probabilities of the weather scenarios, for example to emphasise the less favourable outcomes in the risk averse case. However, it would be more useful to have this explicit in the algorithm.
9. References


Ford, L. R., Jnr (1956) *Network Flow Theory*. Rand Corporation, California, USA


