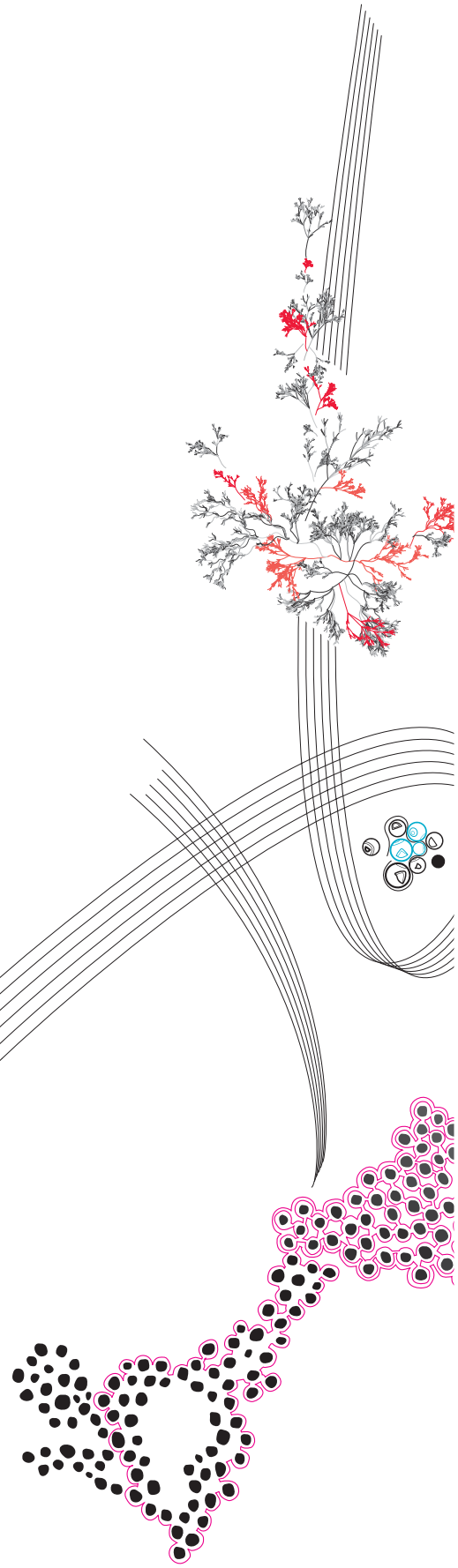


BSc Thesis Applied Mathematics

Optimal Race Strategy for Solar Team Twente Using Approximate Dynamic Programming

Lotte Gerards

Supervisors: Dr. Ir. J. Goseling and Ir. A. Brentjes

June, 2019

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

This bachelor thesis, *Optimal Race Strategy for Solar Team Twente Using Approximate Dynamic Programming*, was written as a completion of the bachelor program Applied Mathematics at University of Twente. During the last months I have learned a lot, not only about constructing a mathematical model, but also about working alone on such a big assignment. I have been working hard on this thesis and the end result is now lying before you.

I would like to thank both my supervisors, Jasper Goseling and Antoni Brentjes, Jasper Goseling for formulating this Bachelor Assignment and guiding me along the way and Antoni Brentjens for giving me insights in the wishes of the Solar Team and helping me when I got stuck.

Furthermore, I would like to thank some of my friends. Thank you Lucas Jansen Klomp for discussing ideas with me and checking this paper. I would like to thank Femke Boelens for keeping me motivated to go to the university every morning to work on my assignment. Wisse van der Meulen and Leander van der Bijl thank you for helping me with my Matlab problems, Wisse for solving my random mistakes and Leander for helping me to understand why Matlab gave an error. Also, I want to thank Annemarie Jutte, Jesse van Werven, Jarco Slager, Sven Dummer and Tessa Rutjens for the distraction between working on this thesis.

Lastly, I also would like to thank Nienke Gerards for checking this report and both her, my parents and my brother for always supporting me.

I hope you enjoy your reading.

Lotte Gerards
Enschede, June 28, 2019

Optimal Race Strategy for Solar Team Twente Using Approximate Dynamic Programming

Lotte Gerards

June, 2019

Abstract

Every two years Solar Team Twente competes in the Bridgestone World Solar Challenge. Their goal is to finish in first place and to reach this goal, an optimal race strategy is needed. Both the uncertainty in the weather forecasts and the risk the Solar Team wants to take should be taken into account when obtaining the strategy. In order to achieve this, the race is modelled as a Markov Decision Process. The uncertainty in the weather forecasts is included by using Affine Kernel Dressing. Also, an exponential utility function is used to incorporate risk sensitivity into the model. The model is implemented using Approximate Dynamic Programming and tested for a simplified case.

Keywords: Markov Decision Process, Approximate Dynamic Programming, Affine Kernel Dressing, race strategy

1 Introduction

In 1987 the first edition of the Bridgestone World Solar Challenge in Australia was held, set up by Hans Tholstrup and Larry Perkins. Nowadays, this race takes plays every two years and universities from all over the world participate in it. The cars travel more than 3000 kilometers from Darwin to Adelaide, merely on the energy of the sun. Moreover, the newest developments concerning vehicle technology and electric cars are deployed. The participants can compete in three different classes, namely the the Challenger Class, the Cruiser Class and the Adventure Class. [2] [3]

Solar Team Twente started competing in the Bridgestone World Solar Challenge in 2005 and has participated ever since. Their best result was a second place in 2015. This year they will again take part in the Challenger Class with their car The Red E. The aim is to "challenge the future of sustainable mobility by using the power of the sun as a primary source of renewable energy" [12].

In order to obtain the best result possible in the race not only an efficient and aerodynamic car is needed, but also a smart strategy. The strategy consists of choosing the right velocity during the race such that the finish time is minimised. The car could be very fast, but when all energy is consumed, this does not result in a fast finish time. Since the car is powered by the sun, the strategy relies heavily on the weather conditions and forecasts. Moreover, the position in the race could influence the strategy. When the car is driving in second place, the team could take more risk in order to come in first place. On the other hand, when the car is in leading position, this place should only be maintained and less risk can be taken. For this situation it is not important anymore to overtake other cars, but to stay ahead of them without running out of energy. Therefore, a part of this paper will focus on designing a model for calculating a strategy that minimises the finish time of Solar Team Twente and takes into account the risk level.

Due to this risk level and updated weather forecasts, the Solar Team Twente could desire to change their strategy during the race. This means that the calculation of a new strategy should

proceed quickly. The model should be implemented efficiently to meet this requirement of the Solar Team. Consequently, the second point of focus is implementing the model such that the strategy can be changed during the race.

In order to meet the requirements of the Solar Team, a finite horizon Markov Decision Process [8] will be constructed that represents the race from start to finish. Both the model and the details about the car and weather forecasts will be discussed. This model will be implemented using a technique called Approximate Dynamic Programming [8]. By doing so, this paper contributes to finding an optimal strategy for Solar Team Twente. Furthermore, more insight in the application of Approximate Dynamic Programming is given.

In Section 2 previous research done on this topic will be discussed. The construction of the model and the translation of ensemble weather forecasts into a probability distribution are described in Section 3, along with inclusion of risk sensitivity. The implementation using Approximate Dynamic Programming is the subject of Section 4. The implementation is tested in Section 5 for a simplified case.

2 Related work

In the past years, three other bachelor theses were written about the optimal strategy for Solar Team Twente. These were written in 2015 [6], 2017 [11] and 2018 [14].

The first bachelor thesis describes a model that is similar to the model the Solar Team still uses. The strategy is determined using Dynamic Programming, specifically Dijkstra's algorithm. The weather forecasts available for this thesis were in the form of the probability that the value for the sun income will be contained in a certain interval, with a total of three intervals. [6]

The main difference in the next thesis was the availability of ensemble weather predictions. These ensemble weather forecasts were analysed separately and brought together afterwards. With these weather forecasts, the model of the previous bachelor thesis was validated. [11]

The last thesis modelled the race as a Markov Decision Process. Moreover, in the paper it was proposed to use the method of Affine Kernel Dressing to handle the ensemble weather predictions. It also looked at risk sensitivity and included this using an exponential utility function. These three ideas will also be used in this paper. The implementation was done using backwards Dynamic Programming. This approach turned out to be too computationally expensive in practice to be of use. A recommendation was made for the method Approximate Dynamic Programming. [14]

In this paper, a Markov Decision Process will be constructed, comparable to the previous bachelor thesis. The construction will follow Chapter 5 of the book *Approximate Dynamic Programming* by Warren B. Powell [8]. This book also describes a method of solving Markov Decision Processes, called Approximate Dynamic programming. This more efficient method, compared to backwards Dynamic Programming, will be used for the implementation of the model.

Furthermore, we again will take into account the weather in the form of ensemble predictions. These will be processed using the Affine Kernel Method described in the paper *From ensemble forecasts to predictive distribution functions* by Jochen Bröcker and Leonard A. Smith [1], which was also used in [14].

The risk level will be included following the paper *Risk-Sensitive Markov Decision Processes* by Ronald A. Howard and James E. Matheson [4], again also used by [14]. They make use of an exponential utility function to describe the behaviour of a Markov Decision Process when it should act risk sensitive.

3 Model

We will derive a Markov Decision Process in order to calculate the optimal race strategy for Solar Team Twente. Before doing so, we also need take a look at the driving costs of the car, the weather forecasts and including risk into the model.

3.1 Driving costs

The solar car consumes an amount of energy while driving, which is depending on the velocity v. These driving costs $D(v)$ can be calculated by multiplying the time costs $C(v)$ by the power of the solar car $P(v, v_{\text{wind}})$, which is also dependent on the velocity of the wind. In the actual model, the driving costs will be important for calculating the decrease in State of Charge of the solar car, which is the energy that is contained in the motor of the car. The driving costs can be described in the following formula:

$$D(v, v_{\text{wind}}) = C(v) \cdot P(v, v_{\text{wind}}). \tag{1}$$

The time costs are defined to be the time it costs to travel a certain distance and those will be specified for the model in Section 3.4.4. The power of the solar car can be calculated in the following manner:

$$P(v, v_{\text{wind}}) = \eta_m \cdot v \cdot F(v, v_{\text{wind}}),$$

where η_m describes the efficiency of the motor and $F(v, v_{\text{wind}})$ is the total drag force. This force consists of two types of forces, namely the aerodynamic drag and the rolling resistance:

$$\begin{aligned} F(v, v_{\text{wind}}) &= F_{\text{aero}}(v, v_{\text{wind}}) + F_{\text{roll}}(v, v_{\text{wind}}) \\ &= \frac{1}{2}\rho C_D A(v + v_{\text{wind}})^2 + mg(vC_{\text{dyn}} + C_{\text{stat}}). \end{aligned} \tag{2}$$

All the parameters are described in Table 1 together with their values. The variable v_{wind} is assumed to be zero for simplicity. [14]

TABLE 1: Parameters of Equation 2 with description and value.

Parameter	**Description**	**Value**
η_m	Efficiency of the motor	1
ρ	Air density	1.2 kg/m^3
$C_D A$	Air drag coefficient times area	0.11 m^2
m	Mass of the car	230 kg
g	Gravitational acceleration	9.81 m/s^2
C_{dyn}	Dynamical part of the Rolling resistance	0 s/m
C_{stat}	Static part of the Rolling resistance	0.003683
v_{wind}	Velocity of the wind	0 m/s

3.2 Weather forecasts

A very important factor in finding the optimal strategy for the Solar team is the uncertainty in the weather forecasts. The sun provides the fuel for the car and therefore influences with which velocity the car can drive. The Solar Team deals with ensemble weather predictions and from these a suiting probability distribution on the sun income will be established using Affine Kernel Dressing. The information in this section will be used in the actual to calculate the increase in State of Charge.

3.2.1 Ensemble weather predictions

The weather forecasts that are available for the Solar Team are in the form of ensemble predictions. This is a set of d different weather forecasts. In our situation, we are interested in the sun income. For a specific place n and time t, these forecasts for the sun income can be presented in a vector of length d:

$$\mathbf{x}(n,t) = [x_1(n,t), ..., x_d(n,t)],$$

with each $x_j(n,t)$ for $j \in \{1, ..., d\}$ a different forecast. Every prediction of the ensemble, called a member, is a possible scenario for the weather at this place and time. They are calculated using multiple models that deviate slightly from one main model and by varying parameters within these models. Therefore, together they give insight in the uncertainty in the weather forecasts. This results in weather forecasts that start close together and when time passes, they start to part from each other. [4]

3.2.2 Affine Kernel Dressing

In the paper *From ensemble forecasts to predictive distribution functions* by Jochen Bröcker and Leonard A. Smith the method Affine Kernel Dressing (AKD) is introduced to construct a suitable probability distribution for the ensemble [1]. They propose the following probability density function for an ensemble $\mathbf{x}$:

$$p(y; \mathbf{x}) = \frac{1}{d \cdot \sigma} \sum_{j=1}^{d} K\left(\frac{y - z_j}{\sigma}\right),$$

where σ stands for the standard deviation, y is the so called verification, that is, the outcome, and z_i is a member of a new ensemble obtained from the original ensemble $\mathbf{x}$. The kernel K will be the standard Gaussian density as proposed in the paper for simplicity. These quantities are given by the following three formulas:

$$\begin{aligned}
\sigma^2 &= h_{\mathcal{S}}^2(s_1 + s_2 v(\mathbf{z})), \\
z_j &= a x_j + r_2 m(\mathbf{x}) + r_1, \\
K(\xi) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\xi^2},
\end{aligned}$$

with constants $\{a, s_1, s_2, r_1, r_2\}$. In these formulas we have the mean $m(\mathbf{x})$ and variance $v(\mathbf{x})$ of the ensemble and Silverman's factor defined by:

$$\begin{aligned}
m(\mathbf{x}) &= \frac{1}{d} \sum_{j=1}^{d} x_j, \\
v(\mathbf{x}) &= \frac{1}{d} \sum_{j=1}^{d} (x_j - m(\mathbf{x}))^2, \\
h_{\mathcal{S}} &= 0.5 \cdot \left(\frac{4}{3d}\right)^{\frac{1}{5}}.
\end{aligned}$$

In order to determine the parameters $\{a, s_1, s_2, r_1, r_2\}$ a scoring rule can be used. The scoring rule measures the performance based on a set of forecasted weather and actual weather observation, together called a training set. A lower scoring rule value yields a better forecast. For this

we will use what is called the empirical score combined with the scoring rule called the Ignorance Score, for a sequence of probability density functions $p_m(y; \mathbf{x})$ with corresponding verification Y_m,

$$S_N(\{a, s_1, s_2, r_1, r_2\}) = \frac{1}{M} \sum_{m=1}^{M} -\log(p(Y_m; \mathbf{x})).$$

This score can be minimised with respect to the parameters using a training set.

3.2.3 Increase in State of Charge

The energy stored in the battery, which is called the State of Charge, of the Solar car will not be the total energy coming from the sun. Suppose we have a predicted sun income y, then we can calculate the predicted increase in State of Charge $\hat{s}$ due to this sun income. For time costs $C(v)$ we have:

$$\hat{s}(y) = \eta_c \cdot A_c \cdot C(v) \cdot y. \tag{3}$$

The values of the parameters for the solar car of Solar Team Twente in this equation are given in Table 2, together with their description and value.

TABLE 2: Parameters of Equation 3 with description and value.

Parameter	**Description**	**Value**
η_c	Efficiency of a Solar cell	0.24
A_c	Total area of the Solar cells	4 m^2

3.3 Risk level

The Solar Team inquired to be able to transition between different strategies. Therefore, it is needed to include risk sensitivity into the model. The Solar Team could choose a risk seeking, risk neutral or risk averse strategy. The paper *Risk-Sensitive Markov Decision Processes* written by Ronald A. Howard and James E. Matheson describes a way to represent these different attitudes towards risk in a maximisation problem [5]. They introduce an exponential utility function, to include the risk level. For a minimisation problem this utility function will have the following form:

$$u(R) = -(\text{sgn } \gamma)e^{\gamma R}.$$

The R in the equation represents the time it will still take to finish, also called the residual time, and γ is the risk aversion coefficient. By varying this coefficient, different risk levels are represented.

A negative γ implies a risk seeking attitude. The utility function will take the form of a convex function. In this case, the lower values of R will have a higher contribution to the total expected value of R. A positive γ, on the other hand, represents a risk averse attitude. Therefore, the utility function is a concave function. The lower values of R will have less influence on the total expected value of R than in the previous case. When γ is zero, the strategy of the Solar Team would be risk neutral. In Figure 1 two examples of the utility function are given for $\gamma = 1$ and $\gamma = -1$ together with the case of $\gamma = 0$

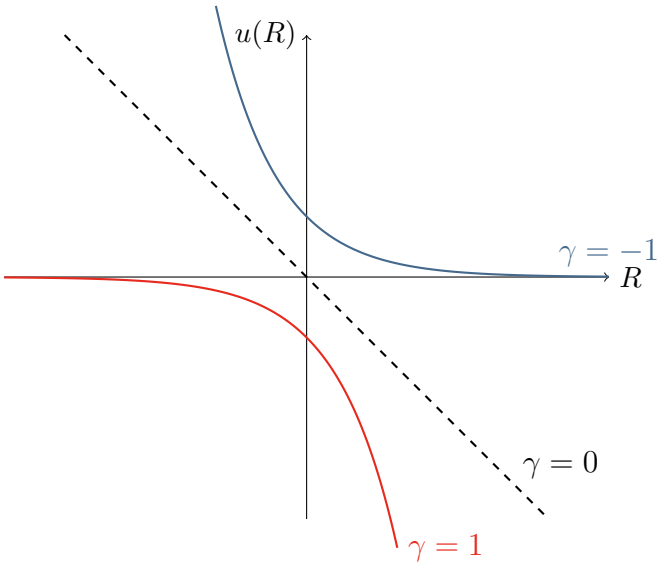


FIGURE 1: Example of utility function for $\gamma = 1$ and $\gamma = -1$.

To return from the value of the utility function w to the actual residue time, we need the inverse of the utility function. This is given by

$$u^{-1}(w) = \frac{1}{\gamma} \log(-(\text{sgn}\,\gamma)w).$$

3.4 Markov Decision Process

In the book *Approximate Dynamic Programming* by Warren B. Powell a notational style is proposed, which will be followed when constructing this finite horizon Markov Decision Process [8]. We will first place N nodes on the road of the race of length L, where we have node 1 at the beginning of the race and node N at the end. This means the race is divided into $N-1$ segments. The current node we are at will be called node n, where $n \in \{1, ..., N\}$. For these nodes the following process is defined:

- *State variable*: The time t_n already passed and State of Charge of the Solar car s_n both at node n.
- *Decision variable* : The chosen velocity v_n during the segment between node n and $n+1$.
- *Exogenous information*: The increase in State of Charge $\hat{s}_{n+1}$ due to the incoming sun during the segment between node n and $n+1$.
- *Transition function*: The function describing the probability of going from time t_n and State of Charge s_n to time t_{n+1} and State of Charge s_{n+1}, depending on v_n and $\hat{s}_{n+1}$.
- *Value function*: The minimum expected time needed to reach the finish, also called the residual time, $R_n(t_n, s_n)$, at node n given that the current time is t_n and the State of Charge is s_n.

In Figure 2, a visual presentation of the model can be seen. The five elements of the Markov Decision Process described above will now be explained in more detail.

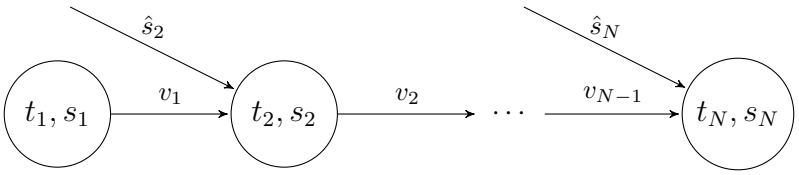


FIGURE 2: Visual presentation of the Markov Decision Process.

3.4.1 State variable

The state of the Markov Decision Process is given by the time and the State of Charge of the solar car at node n. The maximum increase in time during a segment would be the time it takes to drive the whole length of this segment with the minimum velocity v_{min}. Also, the minimum increase in time would be the time it takes to to drive the whole length of the segment with the maximum velocity v_{max}. Therefore, we have that

$$t_n \in \left[t_{n-1} + \frac{L/(N-1)}{v_{\text{max}}}, t_{n-1} + \frac{L/(N-1)}{v_{\text{min}}} \right].$$

For the State of Charge the lower bound would be zero, since a State of Charge below zero is physically impossible. Moreover, it cannot exceed the maximum capacity of the battery. Therefore, we may write $s_n \in [0, s_{\text{max}}]$, in which s_{max} is the maximum capacity of the battery.

3.4.2 Decision variable

At the beginning of every segment a velocity will be chosen for this specific segment. In total $N-1$ decisions will be made, where the last decision takes place at node $N-1$. On the road some minimum and maximum velocities hold, which we will call v_{min} and v_{max}, which can possibly depend on node n. The velocity will be chosen discretely for simplification. This leads to the following domain for the velocity chosen at node n,

$$\mathcal{V}_n = \left\{ v_{\text{min}} + \delta_v \cdot k \;\middle|\; k \in \left\{ 0, 1, ..., \frac{v_{\text{max}} - v_{\text{min}}}{\delta} \right\}, s_{n+1} \geq 0 \right\},$$

with δ_v the step size and s_{n+1} is the State of Charge on the next node, which will be described in section 3.4.4. When the State of Charge is equal to zero, we will assume that the car can still move with the minimum velocity. We will choose v_n such that $v_n \in \mathcal{V}_n$. Furthermore, we will assume that the time it takes to transition between different velocities is negligible.

3.4.3 Exogenous information

At node $n+1$ it becomes clear how much energy from the sun has been stored in the battery of the solar car during the segment between node n and node $n+1$. This information will be named $\hat{s}_{n+1}$ and comes from the probability distribution for the sun income. Therefore, this is a random variable and it gives the transition function described in the next section.

3.4.4 Transition function

The transition function describes the transition from one state to another. First we will consider how the state changes when a certain velocity is chosen. Moreover, we will describe the probability of going from one state to another.

The state variable time will only depend on the time costs of a segment, $C(v)$. The time costs can be calculated using the chosen velocity and the length of a segment, which is the total length of the race L divided by the number of segments $N-1$. Therefore, the transition from state n to state $n+1$ of time becomes known directly after the velocity is chosen. The relation between these quantities can be described as follows:

$$t_{n+1} = t_n + C(v_n), \tag{4}$$

with

$$C(v_n) = \frac{L/(N-1)}{v_n}. \tag{5}$$

These time costs are the same as the ones described in Section 3.1, where the driving costs of the solar car are described.

It is certain with which amount the State of charge will decrease due to driving a segment. However, the battery of the solar car will also be charged by the energy of the sun. Due to the uncertainty in the weather forecasts, the incoming energy of the sun will be a random variable. The State of Charge of the solar car will therefore both depend on velocity of the car and the incoming sun, which will depend on the current time and place. By using this exogenous information, the new State of Charge can be described in the following way:

$$s_{n+1} = s_n + \hat{s}_{n+1} - D(v_n), \tag{6}$$

where $\hat{s}_{n+1}$ is described in section 3.4.3 and $D(v)$ is described in Section 3.1. Since the State of Charge cannot be less than zero, we need to choose a velocity such that this value is non-negative. Also, When s_{n+1} exceeds the maximum capacity of the battery s_{max}, it will take the value of the maximum.

From the ensemble weather predictions the probability distribution of the sun income can be calculated as described in Section 3.2. We obtain the following probability density function for the random variable $\hat{s}_{n+1}$, where $\hat{s}_{n+1}$ can be obtained from y by Equation 3:

$$p_{\hat{s}_{n+1}}(y|\mathbf{x}, n, t) = \frac{1}{d\sigma} \sum_{j=1}^{d} K\left(\frac{y - z_j(n,t)}{\sigma}\right).$$

3.4.5 Value function

The aim of this model is to minimise the expected finish time of the solar car. This should be represented in the value function. We define $R_n(t_n, s_n)$ as the expected time it takes to finish from the beginning of segment n when it is known that the current passed time is t_n and the State of Charge is s_n. This will be called the residual time. This value function is given by:

$$R_n(t_n, s_n) = \min_{v_n \in \mathcal{V}_n} \left(C(v_n) + \mathbb{E}\left\{R_{n+1}(t_{n+1}, s_{n+1}) \mid t_n, s_n, v_n\right\}\right). \tag{7}$$

In this function we have $C(v_n)$ as the time it costs to cover the segment between node n and $n+1$, described in Equation 5. Then there is the expected value of the residual time at node $n+1$. This expected value is dependent on the transition probabilities:

$$\mathbb{E}\{R_{n+1}(t_{n+1}, s_{n+1}) \mid t_n, s_n, v_n\} = \int_0^{y_{\text{max}}} p_{\hat{s}_{n+1}}(y|\mathbf{x}, n, t_n) R_{n+1}(t_{n+1}, s_{n+1}) \mathrm{d}y, \tag{8}$$

with the density function $p_{\hat{s}_{n+1}}$ and the state (t_{n+1},s_{n+1}) as described in Section 3.4.4. The value y_{max} can be found by:

$$y_{\text{max}} = \frac{s_{\text{max}}}{\eta_c \cdot A_c \cdot C(v_n)}.$$

The risk level is not yet included in Equation 7. The utility function described in Section 3.3 is utilised in order to construct the following value function:

$$u(R_n(t_n, s_n)) = \min_{v_n \in \mathcal{V}_n} \left(e^{\gamma C(v_n)} \mathbb{E}\left\{ u(R_{n+1}(t_{n+1}, s_{n+1})) | t_n, s_n, v_n \right\} \right), \qquad \text{for } \gamma \neq 0$$
$$R_n(t_n, s_n) = \min_{v_n \in \mathcal{V}_n} \left(C(v_n) + \mathbb{E}\left\{ R_{n+1}(t_{n+1}, s_{n+1}) | t_n, s_n, v_n \right\} \right), \qquad \text{for } \gamma = 0,$$

where we separate the risk neutral case, when $\gamma = 0$, and the risk seeking or risk averse case, when $\gamma \neq 0$.

4 Implementation of the model

For the implementation of the model, a method called Approximate Dynamic Programming is chosen. The basics of this method are explained in this section. Furthermore, the discretization of the model is described. The model will be inplemented in MATLAB.

4.1 Approximate Dynamic Programming

When solving a Dynamic Program, often backwards recursion is used. However, some problems are too large to solve in this way. This has to do with the three curses of dimensionality, described in [9]. The first curse is that the state space is very large and therefore it is computational expensive to enumerate over all the states. Expectations that are impossible to compute exactly are the second problem. Thirdly, some problems have a decision variable in the form of a vector, which makes the problem hard to solve.

The method Approximate Dynamic Programming (ADP) tries to solve these curses of dimensionality. This method solves Dynamic Programming models in an efficient way and is therefore very useful for large problems.

4.1.1 Algorithm for Approximate Dynamic Programming

With ADP the idea is that you move forward over the decision moments, so forward over the nodes placed on the length of the race in our case, instead of backwards, as is often done when solving a Dynamic Program.

Since the future values of the value function are unknown when stepping forward in space, they need to be approximated. We start with an initial value of the value function for every possible state and place. These are stored in what is called a look-up table, which we call $\bar{R}$. The initial look-up table needs to be updated. This is done by going over different possibilities of states generated by the probability distribution on which they depend. These are called sample paths. In our case a sample path would be a set of values for the sun income, one for each states we end up. These are sampled from the probability density function obtained using Affine Kernel Dressing. Note that since only the State of Charge changes due to the sun income, the state variable time does not depend on the sample path that is taken.

A total of K samples paths will be generated and, therefore, we have K iterations. For every sample path the best decision for every decision moment is chosen and the value $\hat{r}_n^k$ of the value function is calculated in the following way for node n and sample path k, with $k \in \{1, ..., K\}$:

$$\hat{r}_n^k = \min_{v_n \in \mathcal{V}_n} \left(C(v_n) + \mathbb{E}\left\{ \bar{R}_{n+1}^{k-1}(t_{n+1}, s_{n+1}) \;\middle|\; t_n^k, s_n^k \right\} \right).$$

In order to calculate $\hat{r}_n^k$, the values in the look-up table are used. The calculation of the expectation will be described in Section 4.4. The values in the look-up table will be updated using the value

calculated from the minimisation problem:

$$\bar{R}_n^k(t_n^k, s_n^k) = (1 - \alpha_k)\bar{R}_n^{k-1}(t_n^k, s_n^k) + \alpha_k \hat{r}_n^k,$$

with the step size being described by α_k, which could depend on the k. This step size is equal to 1 when we can calculate the transition probabilities exactly [8]. After the update, the next state of the system will be calculated using the sample path in Equations 4 and 6, where is described how the state changes by the chosen velocity and the sun income. We will use Algorithm 1 below to approximate the value function for every possible state:

Algorithm 1: Generic Approximate Dynamic Programming algorithm [9].

Initialisation:

Step a. Initialise the look-up table $\bar{R}$ for all states;

Step b. Choose the initial state (t_1^k, s_1^k) from which every iteration will start;

Step c. Choose the number of iterations K;

Approximate Dynamic Programming:

for *k = 1, 2, ..., K* **do**

  for *n = 1, 2, ..., N* **do**

    Step a. Generate sample ω_n^k;

    Step b. Solve

$$\hat{r}_n^k = \min_{v_n \in \mathcal{V}_n} \left(C(v_n) + \mathbb{E}\left\{ \bar{R}_{n+1}^{k-1}(t_{n+1}, s_{n+1}) \,\middle|\, t_n^k, s_n^k \right\} \right)$$

    and store the decision for which the minimisation problem is solved as v_n^k ;

    Step c. Update the look-up table $\bar{R}$ by

$$\bar{R}_n^k(t_n^k, s_n^k) = (1 - \alpha_k)\bar{R}_n^{k-1}(t_n^k, s_n^k) + \alpha_k \hat{r}_n^k;$$

    Step d. Calculate next state (t_{n+1}^k, s_{n+1}^k) using v_n^k and ω_n^k;

  end

end

This algorithm could be able to solve the curses of dimensionality. However, there are still some parts of the algorithm that need to be explained in more detail, for example the method for generating sample paths and the choice for the initialisation of the look-up table. Moreover, the algorithm works for a discrete model while in our case, the state space of the model is still continuous. Lastly, the calculation of the expectation of the residual time in the next state needs to be explained. These topics will all be discussed in the other parts of this section.

4.1.2 Generating sample paths

For the Approximate Dynamic Programming method, sample paths must be created every iteration. When a variable is uniformly distributed, the sample path is created easily by using the MATLAB function UNIFRND. However, as described in Section 3.2, the ensemble weather predictions will be combined in a probability density function using Affine Kernel Dressing. The probability density function is built from a sum of standard Gaussian distributions.

Suppose we now have a random variable Y with a given cumulative distribution function $F_Y(y) = P(Y \leq y)$. We know that the outcome of the cumulative distribution must have a value between 0 and 1 and that it is uniformly distributed following $U(0, 1)$ [8]. Therefore, if we can

calculate the inverse of this function, we have that:

$$Y = F_Y^{-1}(U(0,1)).$$

Then by generating a random number following a uniform distribution between 0 and 1 and filling this into the equation above, we have generated a random number distributed by the probability function $F_Y(y)$.

Our problem is to find the cumulative distribution function from the density function. For this the density function of the random variable $\hat{s}_{n+1}$ will be integrated and by the theorem of Fubini [13] we can rewrite this as:

$$\begin{aligned} P(\hat{s}_{n+1} \leq y) = &\int_{-\infty}^{y} \frac{1}{d \cdot \sigma} \sum_{j=1}^{d} K\left(\frac{\bar{y} - z_j}{\sigma}\right) \mathrm{d}\bar{y} \\ = &\frac{1}{d} \sum_{j=1}^{d} \int_{-\infty}^{y} \frac{1}{\sigma} K\left(\frac{\bar{y} - z_j}{\sigma}\right) \mathrm{d}\bar{y}. \end{aligned}$$

Note that the expression inside the sum of the last line is exactly the cumulative distribution function of a normal distribution with mean z_j and standard deviation σ, which we will call $P_j(Y \leq y)$. Therefore,

$$P(\hat{s}_{n+1} \leq y) = \frac{1}{d} \sum_{j=1}^{d} P_j(Y \leq y).$$

We will not take the inverse of this equation, as described above. Instead, a random number, uniformly distributed between 0 and 1, will be generated and the corresponding value for y will be calculated numerically, which does the same as when using the inverse function.

4.2 Initialisation

In order for the value function in the algorithm to converge faster, a smart choice for the initialisation of the look-up table is required. Currently, the Solar Team achieves its optimal race strategy by using a form of Dijkstra's algorithm. This method is used to maximise the State of Charge for every possible place and time. The bachelor thesis about the optimal strategy for the Solar Team written in 2015 [6] gives a detailed description of this method. A similar approach could be used to determine the starting point for Approximate Dynamic programming, except that we are interested in minimising the residual time.

We start by dividing the race in different segments, similar to what we did before. For every segment a velocity can be chosen from a finite, discrete set of velocities. The uncertainty in the weather is not included during the calculation. Instead the mean of the ensemble weather forecasts is used (see Section 3.2). A network of states can be created by looking at the states that can be reached driving the different velocities. In Figure 3, an example of such a network can be seen with 2 segments and as possible velocities $\{v_n^a, v_n^b, v_n^c\}$. Every state gives information about the State of Charge and the time that has passed. We see in Figure 3 that at the end of a segment we have three possible states, denoted by $(1), (2), (3)$. Every directed edge in the network has the weight of the time it costs to travel between the nodes it connects. There are only edges between nodes that if one node is reachable from the other.

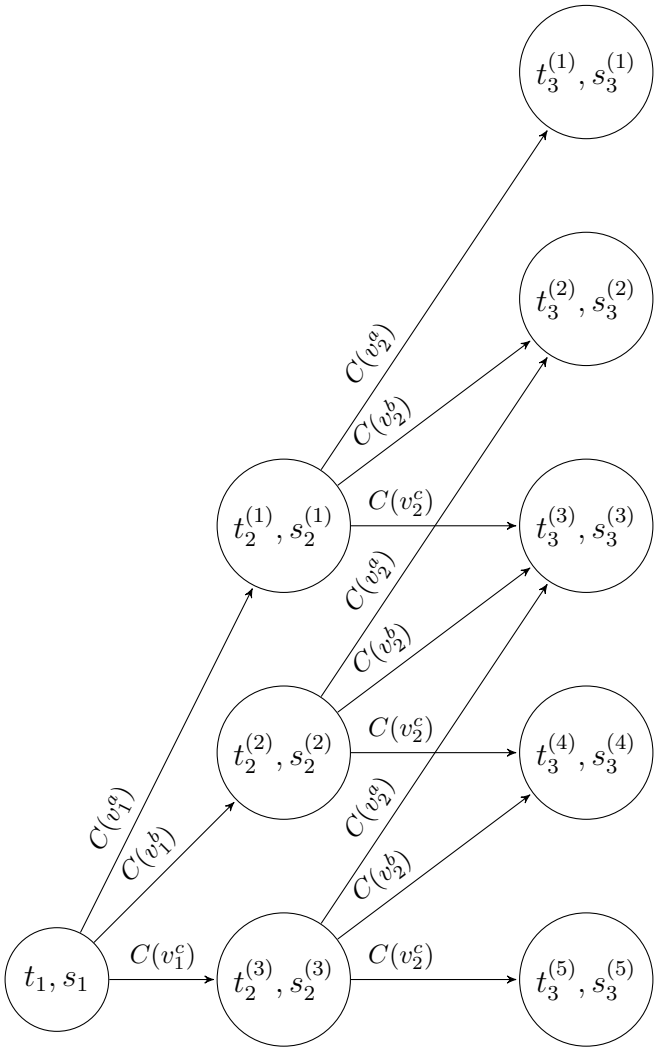


FIGURE 3: Example of a node network for 2 race segments and 3 possible velocities.

When calculating the initialisation of the look-up table, the road network described before could be used. We need an initialisation of the expected time it will cost to reach the finish for every state. Therefore, we will use backward Dynamic Programming to select for every node the best velocity such that the expected finish time is minimised. For this the mean of the ensemble is used as weather conditions. At the end of the race, at node N, all states have as residual time zero. The residual time for the other states can be calculated by the following recursive relation:

$$u(R_n(t_n, s_n)) = \min_{v_n \in \mathcal{V}_n} \left(e^{\gamma C(v_n)} u(R_{n+1}(t_{n+1}, s_{n+1})) \right), \qquad \text{for } \gamma \neq 0$$

$$R_n(t_n, s_n) = \min_{v_n \in \mathcal{V}_n} \left(C(v_n) + R_{n+1}(t_{n+1}, s_{n+1}) \right), \qquad \text{for } \gamma = 0.$$

Note that those relations are similar to the ones in Section 3.4.4, except that we do not need the expectation for the residual time of the next node since there is no uncertainty in the weather. The residual time that is calculated using these equations is stored in the look-up table. We can calculate the next state by

$$t_{n+1} = t_n + C(v_n),$$
$$s_{n+1} = s_n + \mathbb{E}\{\hat{s}_{n+1}\} - D(v_n),$$

where $\mathbb{E}\{\hat{s}_{n+1}\}$ is the mean of the ensemble forecasts. [15]

4.3 Discretization of the model

The model of the Markov Decision Process has continuous states. In order for the model to be implemented using Approximate Dynamic programming, these states need to be discretized. The state space consists of the time passed and the State of Charge at node n. Both depend on the velocity, which is already defined discretely. The State of Charge also depends on a random variable, namely the sun income. We will start by discretizing the state time.

As mentioned in Section 3.4.4, the time changes from node n to node $n+1$ following this equation:

$$t_{n+1} = t_n + \frac{L/(N+1)}{v_n}.$$

Although the velocity is discrete, the time can still take many values and therefore should be divided into multiple intervals, all of length δ_t. Therefore, we will define that t_{n+1} must be within the set:

$$\mathcal{T}_{n+1} = \left\{ t_n + \delta_t \cdot m \;\middle|\; m \in \left\{ 0, 1, ..., \left\lceil \frac{L/v_{\min}}{\delta_t} \right\rceil \right\} \right\}.$$

The maximum finish time will be rounded up to the nearest integer. Whenever the time of node $n+1$ is not contained in this set, it will take the value of the upper bound of the interval in which it is contained. When the value would be rounded down the outcome should be more optimistic than it actually is and this is not preferable.

The State of Charge at the next node is calculated using the following formula, also described in Section 3.4.4,

$$s_{n+1} = s_n + \hat{s}_{n+1} - D(v_n).$$

Similarly as for time, we will define a discrete set in which the State of Charge will be chosen,

$$\mathcal{S}_n = \left\{ \delta_s \cdot r \;\middle|\; r \in \left\{ 0, 1, ..., \left\lfloor \frac{s_{\max}}{\delta_s} \right\rfloor \right\} \right\},$$

with the step size δ_s. Different from the set for time, for the State of Charge we will round down to the lower bound of the interval. When we would round up, it is possible that the State of Charge has reached zero, while the model says it is not empty. This would be problematic during the race.

4.4 Approximation of the expectation

In the method of Approximate Dynamic Programming, a look-up table is used and, as described before, the time and State of Charge needed to be discretized. Therefore, the integral in the expectation of Equation 8 also needs to be approximated. This will be done by using the method of Monte Carlo to approximate the integral by the mean of a sum of randomly generated observations of the State of Charge. These samples are generated as described earlier in Section 4.1.2. For a total of W of these samples, where s_{n+1}^m can be calculated using the m^{th} observation for the sun income, we have that

$$\mathbb{E}\{R_{n+1}(t_{n+1}, s_{n+1}) \mid t_n, s_n, v_n\} \approx \frac{1}{W} \sum_{m=1}^{W} R_{n+1}(t_{n+1}, s_{n+1}^m).$$

By the strong law of large numbers, it can be proven that for enough samples, this sum converges to the integral in Equation 8 [10].

For every iteration k and place n a new set of samples will be generated. The new state obtained from these samples will not be discretized, but instead the value of the value function will be calculated by means of linear interpolation.

5 Results

The behaviour of the model will be tested for a simple case. Throughout this section, we will use the values described in Table 3, unless stated otherwise.

TABLE 3: Parameters used for simplification with description and value.

Parameter	Description	Value
v_{min}	Minimum velocity	1/2
v_{max}	Maximum velocity	1
t_{min}	Minimum time	0
t_{max}	Maximum time	L/v_{min}
s_{min}	Minimum State of Charge	0
s_{max}	Maximum State of Charge	1
δ_s	Step size State of Charge	0.05

For simplification, we would like to have the velocity discrete such that the time does not have to be rounded up. In order to achieve this we will first make the segments all of integer length. Then we choose a δ_t of $1/4$, which leads to the following set of velocities (by using Equation 5):

$$\mathcal{V}_n = \left\{ \frac{1}{2}, \frac{4}{7}, \frac{2}{3}, \frac{4}{5}, 1 \right\}.$$

The driving costs for a segment of length 1 for these velocities will be equal to the velocity squared. The possible sun income is also simplified. We will look at a set of three possibilities for sun income per time unit, namely $\{0, 1/4, 1/2\}$. Each of these sun incomes occurs with probability 1/3. For simplicity the sun income is equal to the increase in State of Charge. The initial state (t_1, s_1) from which every iteration of Approximate Dynamic Programming starts is equal to $(0, s_{\text{max}})$.

In Section 3.4.1 it was mentioned that the State of Charge should not become lower than zero. This has been taken into account when implementing the model by adding extra costs when this happens. These costs are equal to the time it takes to drive that segment with the minimum velocity together with the expected residual time of the next state that is reached by the minimum velocity.

For Approximate Dynamic Programming a step size α_k must be chosen with which the values in the look-up table will be updated. In this situation, α_k will be equal to 1 for every iteration k, since an accurate approximation of the expectation is obtained from the Monte Carlo method. Unless stated otherwise, the initialisation method using Dynamic Programming described in Section 4.2 is used for a constant sun income of 1/4. The risk level has not been implemented.

In the following subsections, we will look at the convergence of the look-up table. Furthermore, the optimal strategies for different weather realisations are analysed. Moreover, these strategies are compared to the ones that would be obtained when using as look-up table the initialisation of Dynamic Programming.

5.1 Convergence of the model

A fast convergence of the look-up table is important for the Solar Team since they would like an efficient implementation, such that the strategy can be changed during the race dependent on the current weather forecasts. For this we look at the difference Δ_{k+1} in the values of the look-up table at the end of iteration k and $k+1$. In order to calculate Δ, the Root Mean Square deviation is used, where the sum is taken over the squared difference for every possible state and segment:

$$\Delta_{k+1} = \sqrt{\sum_n \sum_t \sum_s \left(\bar{R}_n^{k+1}(t_n, s_n) - \bar{R}_n^k(t_n, s_n) \right)^2}$$

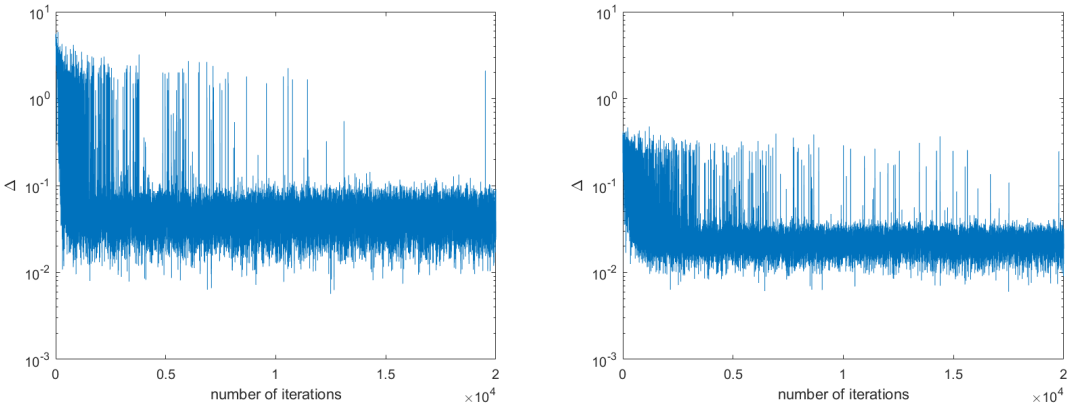
The convergence rate of the model will be analysed using the Root Mean Square deviation for the case of constant weather, for different initialisation and for varying the amount of race segments and Monte Carlo samples.

5.1.1 Verification of the implementation outcome using mean weather forecasts

Before going to a stochastic problem, the implementation is tested using a fixed weather outcome. This results in the look-up table not being updated and therefore already being the expected remaining time to finish for this situation. This is what we would expect the outcome of this implementation to be. Since both calculation methods have the same input of sun income they should also give the same residual time for every state.

5.1.2 Difference in initialisation

When making the problem stochastic, there is a difference in rate of convergence between an initialisation of all states having a residual time of zero or using the outcome of dynamic programming for the mean of the weather forecasts. This first initialisation was chosen since we are minimising the residual time and therefore states with a low initialised value seem attractive to visit [8]. In Figure 10b the difference in convergence rate can be seen for the two possible initialisation when 20,000 iterations are used. This outcome is for a race length of 10 and number of nodes 11. The number of samples used for the Monte Carlo method is 5,000.



(A) Convergence rate for initialisation with zeros.

(B) Convergence rate for initialisation with Dynamic Programming.

FIGURE 4: Converges rate of two different initialisation for 20,000 iterations.

For the zero initialisation in Figure 4a the Root Mean Square value starts higher but ends up fluctuating above the value around which the Dynamic Programming initialisation in Figure 4b fluctuates. However, after quite some iterations, the Dynamic Programming initialisation still shows more peaks when compared to the zero initialisation. This could indicate that the outcome has not converged yet, although the initialisation has been chosen close to the expected outcome. To see whether the algorithm would actually converge for the initialisation using Dynamic Programming, the convergence rate was tested for 40,000 iterations. The result of this can be seen in Figure 5.

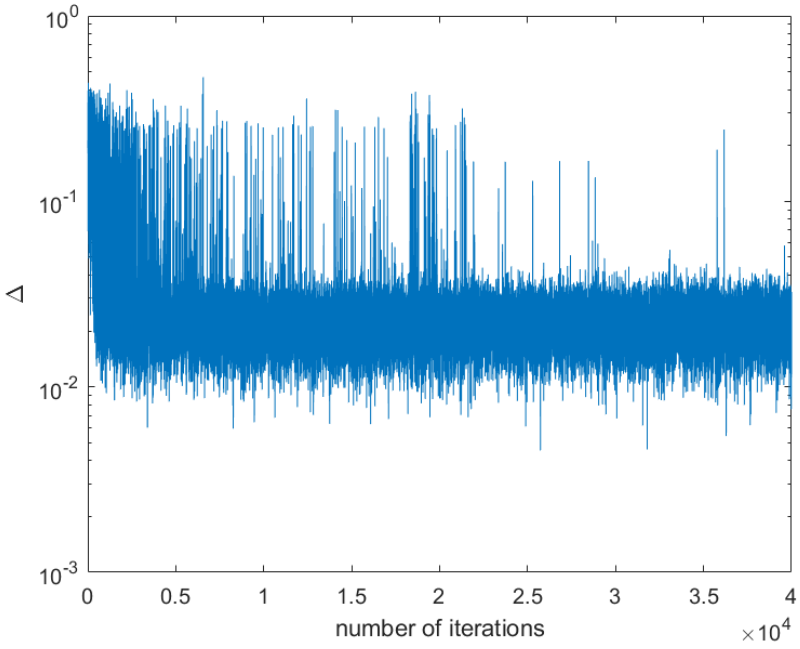


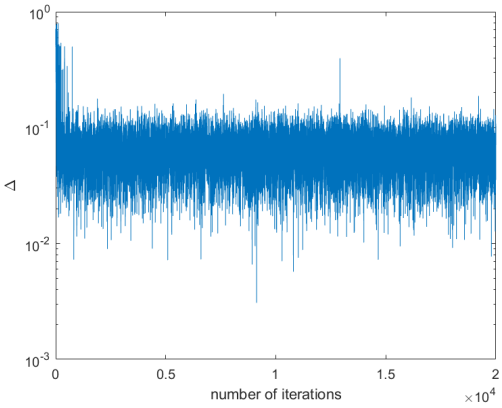
FIGURE 5: Convergence rate using Dynamic Programming as initialisation for 40,000 iterations.

The amount of high peaks decrease when the amount of iterations increases. Therefore, when testing the model in Section 5.2, 40,000 iterations will be used.

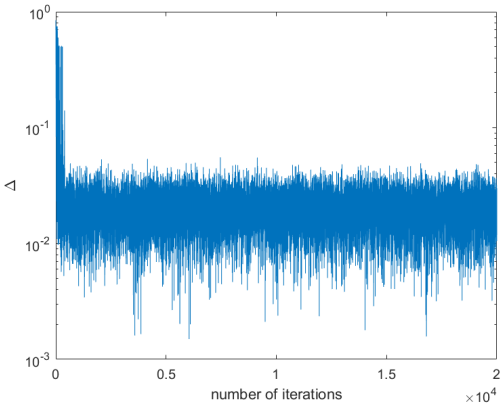
Due to the fact that the convergence rate of the zero initialisation is quite similar to the Dynamic Programming initialisation, we have looked into the converged values for the look-up table of both initialisations. It turns out that the look-up tables differ a lot. For the zero initialisation the expected finish time for the state from which we start is a lot lower, even lower than the time it would take to move the whole race with the maximum velocity. Therefore, these values are not realistic. The difference in value could be caused by the fact that not all have been visited equally often. This results in states being less often updated as other states and therefore could give a weaker approximation of the value function at this state. Since some states are never visited, their value will stay zero. These values are still used to calculate the expectation of the residual time of the next state. This problem is referred to as the exploration versus exploitation dilemma [7]. It could result in a lower expected finish than is actually the case.

5.1.3 Varying the number of nodes and the number of samples

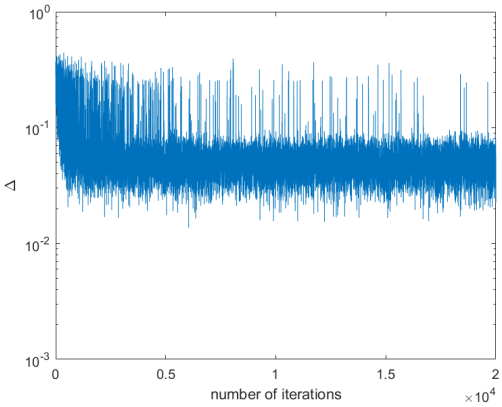
Besides the initialisation, two other factors that can influence the convergence are the number of samples used for the Monte Carlo method and the number of segments. Therefore, we have tested the convergence for either 5 segments or 10 segments and either 1000 samples or 10000 samples. The length of the race is still 10. The convergence rates for these cases can be seen in Figure 6.



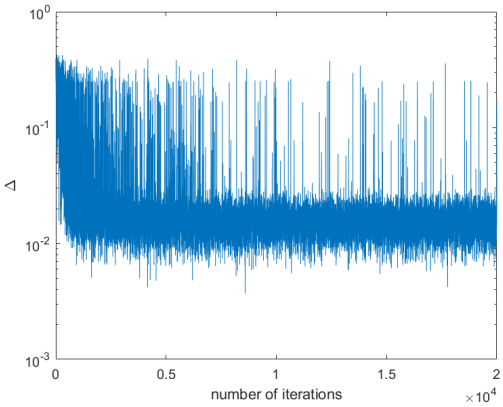
(A) Convergence rate for 5 segments and 1,000 samples.



(B) Convergence rate for 5 segments and 10,000 samples.



(C) Convergence rate for 10 segments and 1,000 samples.



(D) Convergence rate for 10 segments and 10,000 samples.

FIGURE 6: Converges rate when varying number of nodes and number of Monte Carlo samples.

When varying the number of samples, it can be seen that the value to which Δ converges is lower for more Monte Carlo samples. This seems logical, since an increase in the number of samples will result in a better approximation of the expectation. Hence, the residual time is updated by a more accurate value. More samples means less fluctuation in residual time due to the mean of the samples approximating the expected weather forecast better.

When the number of segments is increased, the difference in the Root Mean Square deviation is less for more segments. However, more large peaks be seen for more segments. This could be explained by the fact that there are more possible states and, therefore, the total Root Mean Square will sometimes fluctuate more when a state is reached that is visited less often.

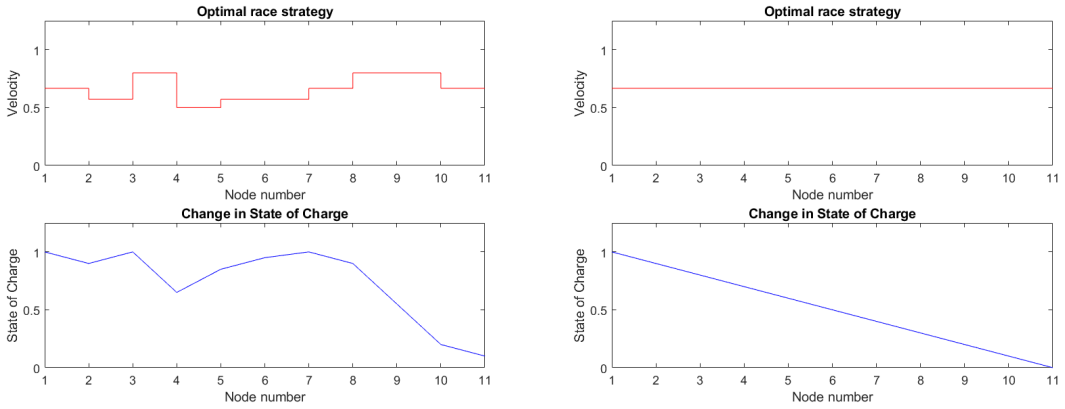
5.2 Strategy for different weather realisations

Eventually, we would like to see a strategy given a specific weather condition. Therefore, it can be interesting to see what strategy is chosen using Approximate Dynamic Programming (ADP) for different weather realisations. This strategy can be compared to the strategy the initialisation with Dynamic Programming would give. For the strategy we will look at both the chosen velocity per segment and the change in State of Charge. The tests will all be done for a race of length 10 with

11 nodes placed evenly over the length of the race. Moreover, the number of Monte Carlo samples is 5,000 and the number of iterations is 40,000. The look-up tables of the initialisation and the ADP method are generated using the weather conditions described earlier.

5.2.1 Same weather as used in initialisation

First, we will look at the given strategy for the mean of the weather forecasts, a constant sun income of 1/4. This is the value for the sun income with which the initialisation is calculated. In figure 7 both strategies can be seen.



(A) Strategy obtained from ADP.

(B) Strategy obtained from the initialisation.

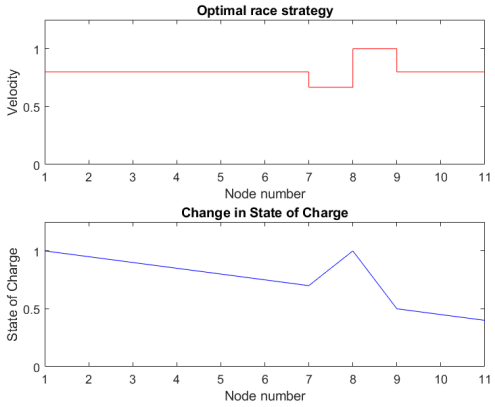
FIGURE 7: Comparison of strategies for constant sun income of 1/4.

A striking difference between the strategies is that the velocities obtained using ADP fluctuate, while the velocities obtained using the initialisation do not. This can also be seen in the change in State of Charge. For the initialisation the State of Charge decreases evenly while for ADP the State of Charge both increases and decreases during the race.

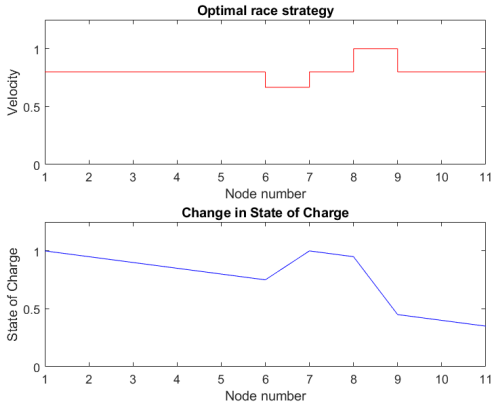
The finish times are 15.5 and 15.0 for the strategies obtained ADP and the initialisation, respectively. The finish time for the ADP method is higher. We expect this to happen due to incorporation of uncertainty in the weather conditions for ADP. For this method, the State of Charge stays higher in the beginning of the race to account for possible bad weather in the future. The initialisation, on the other hand, is optimised for this specific weather condition and therefore will give the fastest time for this sun income. We will also look at the finish times of other weather realisations.

5.2.2 Other possible weather realisations

We will now look at race strategies for weather conditions which different from the mean weather forecast. The first weather condition that is considered is a maximal sun income of 1/2 during the whole race. The strategies for this weather realisation can be seen in Figure 8.



(A) Strategy obtained from ADP.

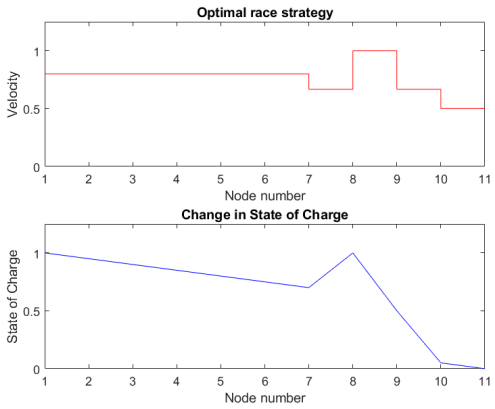


(B) Strategy obtained from the initialisation.

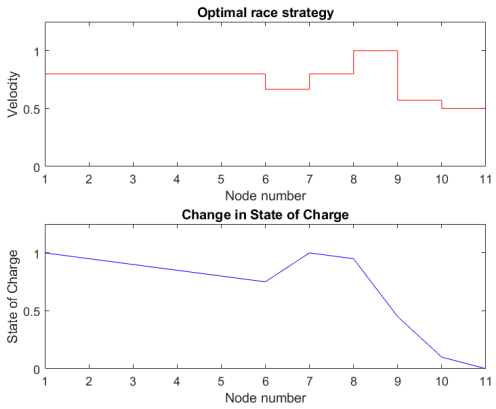
FIGURE 8: Comparison of strategies for constant sun income of 1/2.

The different methods do not give the same strategy, but they both result in a finish time of 12.5. Due to these different strategies, the change in the State of Charge is different for the strategies. It seems logical that better weather conditions result in a faster finish time, which is exactly the case for this model.

Furthermore, we will consider a maximal sun income of 1/2 during the first 8 segments and a minimal sun income of 0 during the last 2 segments. The resulting strategies can be seen in Figure 9.



(A) Strategy obtained from ADP.



(B) Strategy obtained from the initialisation.

FIGURE 9: Comparison of strategies for constant sun income of 1/2 at the beginning of the race and 0 at the end.

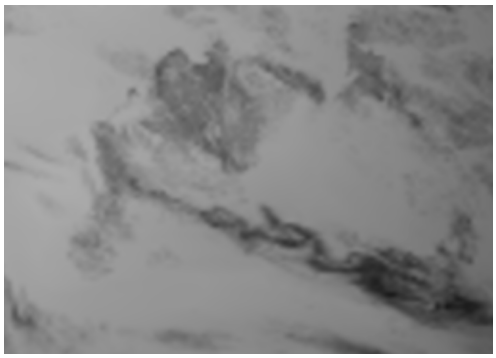
This specific weather realisation was chosen to show that the ADP method can yield a faster finish time when compared to the initialisation. The finish times were 13.5 and 13.75 for the ADP method and the initialisation, respectively. However, note that the State of Charge is smaller for the ADP method at the 10^{th} node. When the State of Charge becomes negative, the car can only drive with the lowest velocity. Both strategies give the lowest velocity for the last segment, but since in the ADP method more energy has already been used, the end time is lower.

These different weather conditions show that the performance of the ADP method differs from the initialisation method that is used. However, it remains a complex matter to compare the

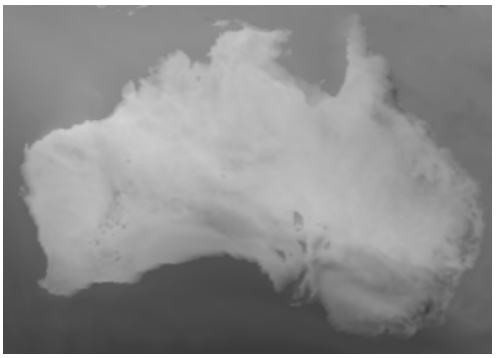
outcome of a stochastic model with a deterministic model. Also, the tests were performed for a simplified situation and therefore a more complicated and realistic situation should also be tested.

5.3 Weather data

When the model would be tested for a more realistic case, we also need realistic weather conditions. Data of the weather forecasts is available from the year 2015. However, this is not an ensemble, but only the mean weather forecast. The data is in the form of multiple grayscale images containing Australia. One image is made every three hours and later every 6 hours. The intensity of a pixel gives the amount of sun income on that place and runs from 0 (black) to 255 (white), respectively 0 to 1530 W/m^2. An example of such a weather prediction can be seen in Figure 10a. In Figure **??**, a picture of the forecast of the temperature can be seen, which is added in order to show where Australia lies in Figure 10as.



(A) Forecast of the sun income on 17-10-2015 6 pm, Dutch time.



(B) Forecast of the temperature on 17-10-2015 6 am, Dutch time.

FIGURE 10: Example of weather forecasts for a specific time.

For the implementation, we still wish to use an ensemble prediction. Therefore we will imitate this by taking different columns of pixels from the picture containing the mean weather forecast, treating each column as a member of the ensemble. We will assume that one column represents the whole length of the race. These columns are not independent, but neither are the members of an ensemble. When we take three columns at a specific place and plot them over time this would result in Figure 11.

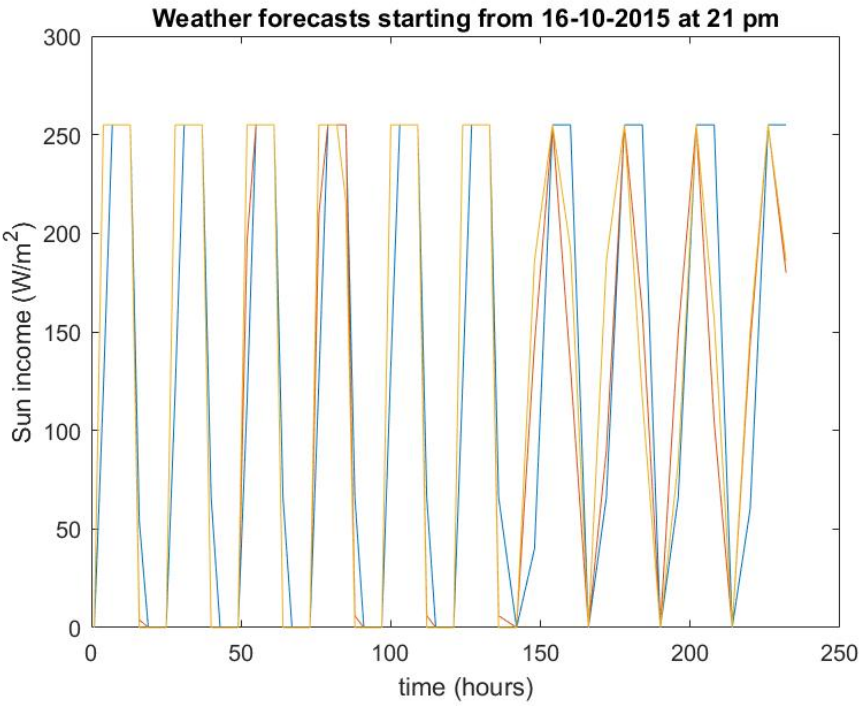


FIGURE 11: Ensemble of three columns over time.

Since the time between pictures is 3 or 6 hours, linear interpolation is applied. Also, for place linear interpolation is used. From a time of 135 hours, the forecasts were done every 6 hours.

We see that the lines stay close together over the whole time, although slightly closer in the beginning than in the end. However, in an actual ensemble, the lines would part from each other over time. Also, the peaks are truncated due to the interpolation. These peaks could be approximated using some sort of sine function. On long term we cannot treat this as an ensemble forecast, but for time close by it could be used for testing the model.

6 Conclusion

In this paper, the stratgy for the race of the Bridgestone World Solar Challenge was modelled using a Markov Decision Process in order to obtain the optimal race strategy for the Solar Team Twente. Using a probability distribution for the sun income obtained from the Affine Kernel Dressing method, the model takes the uncertainty in the weather into account. Furthermore, the risk sensitivity was included using a exponential utility function where a risk aversion coefficient can be varied to choose the risk level. This model is able to satisfy the requirements of Solar Team Twente after implementation.

In order to implement the model efficiently the method of Approximate Dynamic Programming was used. This algorithm was initialised using Dynamic Programming with as sun income the mean weather forecasts, a method similar to the one the Solar Team uses currently. The implementation was tested for a simplified situation. The algorithm eventually converges to a strategy.

The strategy obtained from Approximate Dynamic Programming was compared to the one from the initialisation for different weather realisations. This comparison yielded varying results when looking at the finish time. Therefore, the algorithm should be tested for a more realistic situation.

7 Discussion

Due to time restrictions not all requirements of the Solar Team Twente have been fullfilled. Therefore, in this section we will discuss some recommendations for further research.

Firstly, the model and implementation method should be tested on a more realistic situation to obtain a useful strategy for the Solar Team. Currently, the model has only been tested on a simplified situation. A start was made on the implementation for a more realistic case, but this has not been finished in the given time. For this implementation the data of 2015 would have been used to imitate an actual ensemble and with the method of Affine Kernel Dressing a probability density function would have been obtained. Furthermore, the risk has also not been implemented yet and therefore no tests could be done for this part.

The bachelor thesis of last year on this subject mentions that the Dynamic Programming Approach is computationally expensive. This was the main reason for choosing Approximate Dynamic Programming to come to a strategy. However, for the simplified situation, Approximate Dynamic Programming has proven to be not very fast. Therefore, it is advised to test the algorithm on speed. Also, in the paper *Approximate Dynamic Programming by practical examples* by Martijn RK Mes and Arturo PÃl'rez Rivera it is shown that for their example the use of a form of Value Function Approximation, for example using aggregation, resulted in faster convergence. Such a method could also be used for this problem.

When a realistic model and an actual ensemble prediction would be available, it can be interesting to look at each ensemble member as a sample path in Approximate Dynamic Programming. These members are all possible scenarios for the weather, which is exactly what a sample path should also be. The book *Approximate Dynamic Programming* by Warren B. Powell [8] suggests to use real data as sample paths, since then the underlying distribution does not need be known. The behaviour of the strategy can be interesting to research in this case.

Lastly, in the discretization of the model the time was always rounded up, while the State of Charge was always rounded down. When more nodes were placed over the distance of the race, this resulted in the time being rounded up more often and therefore being less accurate every node. This was the reason for choosing the velocities in the simplified model such that time did not need to be discretized. It could be interesting to see whether the strategy would change if we would alternate between rounding time up and down.

References

[1] Jochen Bröcker and Leonard A Smith. From ensemble forecasts to predictive distribution functions. *Tellus A: Dynamic Meteorology and Oceanography*, 60(4):663–678, 2008.

[2] Bridgestone World Solar Challenge. *History*, 2019 (Access date: 2019-06-09). https://www.worldsolarchallenge.org/about_wsc/history.

[3] Bridgestone World Solar Challenge. *Overview*, 2019 (Access date: 2019-06-09). https://www.worldsolarchallenge.org/about_wsc/overview.

[4] ECMWF. The ecmwf ensemble prediction system. 2012.

[5] Ronald A Howard and James E Matheson. Risk-sensitive markov decision processes. *Management science*, 18(7):356–369, 1972.

[6] Guus Kattenbelt, Arwyn Goos, and Tineke School. Optimale rijstrategie afhankelijk van weersvoorspellingen voor 'the red engine'. 2015.

[7] Martijn RK Mes and Arturo Pérez Rivera. Approximate dynamic programming by practical examples. In *Markov Decision Processes in Practice*, pages 63–101. Springer, 2017.

[8] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[9] Warren B Powell. What you should know about approximate dynamic programming. *Naval Research Logistics (NRL)*, 56(3):239–249, 2009.

[10] Sheldon M Ross. *Introduction to probability models*. Academic press, 2014.

[11] Marije Siemann. Risico van een racestrategie voor de red shift. 2017.

[12] Solar Team Twente. *Car History*, 2019 (Access date: 2019-06-09). https://www.solarteam.nl/history/.

[13] Jan Van Neerven and Mark C Veraar. 26 on the stochastic fubini theorem in infinite dimensions. *Stochastic Partial Differential Equations and Applications-VII*, page 323, 2005.

[14] Lotte Weedage. Optimal race strategy for the solar team twente. 2018.

[15] Wayne L Winston and Jeffrey B Goldberg. *Operations research: applications and algorithms*, volume 3. Thomson/Brooks/Cole Belmontˆ eCalif Calif, 2004.