Towards optimal path planning under uncertainty using Monte Carlo Tree Search

(Final report)

Paul Barde*, Yves Briere*, Emmanuel Rachelson* and Caroline P. Carvalho Chanel*
*Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: paul.barde@student.isae-supaero.fr, yves.briere@isae.fr, emmanuel.rachelson@isae.fr, caroline.chanel@isae.fr

***Abstract*—We consider the problem of optimal long-term path planning for off-shore sailing. This domain of application is challenging as it combines uncertain and time varying wind conditions with complex boat's performances that depend on the heading angle, the wind conditions and the sea state. The wind is estimated from forecasts provided by the Global Forecast System (GFS) files[1] and the ship's dynamic is evaluated with a Velocity Prediction Program (VPP). Based on this, we build a state generator that model state transitions. The corresponding decision making problem has a high branching factor and a large decision space. Thus, a Monte Carlo Tree Search (MCTS) with Upper Confidence Bounds for Trees (UCT) algorithm is implemented to find the sequence of headings that minimizes travel time from a point A to a point B.**

I. Context

Historically, the search for optimum tracks in transoceanic crossings, from long-range prediction of wind, has been dealt with the isochrons method [8]. This method requires little computational resources and is therefore adopted by most of the commercial sail-boat racing software. It provides real time trajectory computation, however, it is purely deterministic and does not model the inherent stochasticity of real world sailing. The ultimate objective of this work would be to outperform software based on the isochrons method. This research is part of the IBOAT project and will be applied to an autonomous sailing robot in order to cross the Atlantic ocean [2].

More recently, the problem of trajectory planning for off-shore sailing has been simplified to short-term path planning. In this downgraded problematic, the wind forecasts are not taken into account. This problem has therefore been tackled with new tools. Potential field algorithms [13] or optimization of the distance's time derivative [16] have been implemented to drive the boat along an optimal trajectory. Both studies use a velocity polar that is either generic [16] or based on experience [13]. However, they only deal with deterministic processes and dynamics. Indeed, they do not consider wind fluctuations or uncertainty in the boat's performances. Nevertheless, it should be noted that [13] adapts the trajectory to the measured changes in the local and instantaneous wind.

The stochastic nature of trajectory planning has been investigated for short term sail-boat racing. The probabilistic features are incorporated either in the wind's fluctuations [1] or in the ship's dynamics [6]. On the one hand, [1] consider deterministic boat's dynamics but stochastic wind's changes. They also incorporate the opponent's actions into their decision process. This formulation is solved by applying a stochastic game approach to yacht match racing. On the other hand, [6] focus on a fully observable environment but uncertain boat's dynamics. They derive an infinite horizon Markovian policy using a value iteration algorithm and a total discounted reward. The transition function for the boat's position is a Gamma distribution depending on the boat's leeway angle. However, all these papers differ from the problematic we wish to tackle in their consideration of short period of time and distance range. As a matter of fact, they do not profit from weather forecasts to optimize the boat's route on a larger scale.

Eventually, the work of [11] deals with long-term planning. They rely on weather forecasts and consider different possible weather scenarios to incorporate forecast uncertainty. They solve the constructed problem using dynamic programming recursion and considering that the boat's performances are deterministic.

These works rely on the Markov Decision Process (MDP) framework which is a wide and well documented field [9], [12]. It is a powerful tool to design intelligent agents that progress autonomously, and for a long period of time, in an uncertain environment through unreliable interactions. In our case, since we model stochasticity in both the boat's dynamic and wind conditions, it is difficult in practice to derive realistic transition functions. And therefore to use classical MDP algorithms. As an alternative, we derive a simulator that generates a new state from a previous state and a taken action. This makes sampling possible and enables the exploration of the MDP space by incrementally constructing a sequential decision tree. As the decision space is almost infinite, we need a efficient search algorithm to build an asymmetrical tree.

Monte Carlo Tree Search (MCTS) is a recent, best-first search, framework [3] that is mostly used for online planning [15]. However, several works presented off-line methods to capitalize on the knowledge accumulated during the search [7], [4]. The brightest example being the AlphaGo victory which has been a major breakthrough in the AI community [14].

MCTS is thus a relevant and advisable framework to investigate path planning for autonomous off-shore sail-boats. Indeed, uncertain wind conditions and boat performances give rise to complex stochastic dynamics that may generate an infinity of states. Coupling this to long-term planning, which requires large temporal horizon, induces a gigantic search

[1] http://nomads.ncep.noaa.gov

space.

In this work we first describe the environment and characterize the wind conditions. Then, the dynamics of the agent, which is a sail-boat in our case, are defined. From this, the interactions between the agent and its environment are modelled through a generative model. At this point, we present the MCTS framework and implement a specific algorithm to explore the decision space and produce a prescription P. This prescription is the sequence of actions that minimizes the duration of the agent's course. Finally, the strategy produced by the search is tested and the perspectives of the work are discussed.

II. The environment

A. Generalities on the wind model and weather forecast

We chose the open-source Global Forecast System (GFS) model which is the most convenient one for our application. Because the boat is slow, we picked the one with the highest spatial resolution. Therefore, we have daily 0.25° spatial resolution outputs providing an eight days forecast. The time step of the forecasts is 3 hours. These grids are linearly interpolated to fit a continuous space approach. They are also interpolated in time to match the simulation's time step. From the 400 GB daily output of the GFS we only consider the 10m above water surface wind (AWSW).

B. Modelling the forecast uncertainty

We used the GFS to have an estimate of the encountered wind on the boat's track. The data is provided by NOAA servers[2]. This model covers an eight days horizon but do not give any measure of the uncertainty nor expresses how this uncertainty increases for long term estimations. In order to model the degradation of the model precision with time, we used the Global Ensemble Forecast System (GEFS) which is also provided by NOAA servers[3]. GEFS runs several models from perturbed initial conditions and computes the ensemble spread of the models [18]. As shown in figure 1, the wind speed ensemble spread increases as we look later in the forecasts. This applies also for the wind direction. Hence, in this work we assume that GFS provides a good estimate of the forecast mean value and that GEFS provides a good measure of the uncertainty. Finally, it can be seen that beyond three days the forecasts become rather unreliable.

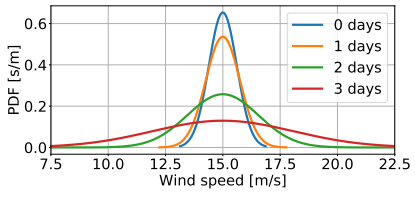


Fig. 1. Gaussian wind speed distribution around GFS mean given GEFS standard deviation

[2]http://nomads.ncep.noaa.gov:9090/dods/gfs_0p25/gfs

[3]http://nomads.ncep.noaa.gov:9090/dods/gens_bc/gens

The characteristics of the GFS and GEFS outputs are summarized in the following table.

	Δt (hours)	Δx (°)	T (days)	**Coverage** (-)	**Data** (-)
GFS	3	0.25	10	Worldwide	10 m AWSW
GEFS	6	1	15.5	Worldwide	10 m AWSW

TABLE I
GFS AND GEFS CHARACTERISTICS SUMMARY

III. The dynamics of the agent

A. The Velocity Prediction Program (VPP)

Based on [13], [6] and [11], we have on a Velocity Prediction Program to determine the velocity of the boat for a given wind magnitude and orientation. To this end, we made use of the Matlab script `gvpp` provided by Gianluca Meneghello[4]. This method is based on the work of [10] and relies on the equilibrium between the aerodynamic and hydrodynamic forces. These forces mostly derive from empirical considerations. The VPP assumes that all the controllable factors (such as the sail configuration) are set so as to maximize the ship speed under the given wind conditions. Hence, from the 10m AWSW, it outputs the velocity $\mathcal{V}$ of the boat in the direction of heading (thus neglecting the drifting of the vessel).

B. Adapting the VPP

a) Speeding up computation: To speed up computations and gain flexibility, these VPP results are fitted with a two-dimensional fifth-order polynomial. The relative error between the fit and the data is less than 3.5%.

b) From VPP to long-term mean speed: In reality, a boat can not cruise at all points of sail α (the angle between heading and wind direction). There is two no-go zones : one for $0° < \alpha < \alpha_{min} = 35°$ (the boat cannot sail directly into the wind) and another for $160° < \alpha < \alpha_{max} = 180°$. To go toward this directions a boat must tack. However, in our case we will consider that the boat keeps a constant point of sail over a long period of time. Hence, the time lost during a tacking manoeuvre is negligible. Following this logic we will assume that the boat can sail towards the no-go zones but with a reduced speed. In practice, we will consider that the speed is equal to the Velocity Made Good (VMG) that is the actual velocity projected toward the goal. In addition, it has been observed that for wind above 19 m/s the boat's performance change little. The resulting velocity polar in figure 2 is coherent with the boat's dynamics. The velocity is maximal for running points of sail but diminishes if the point of sail is greater than α_{max}. Finally, the vessel cannot sail into the wind if this one is too strong.

[4]https://sourceforge.net/projects/gvpp/

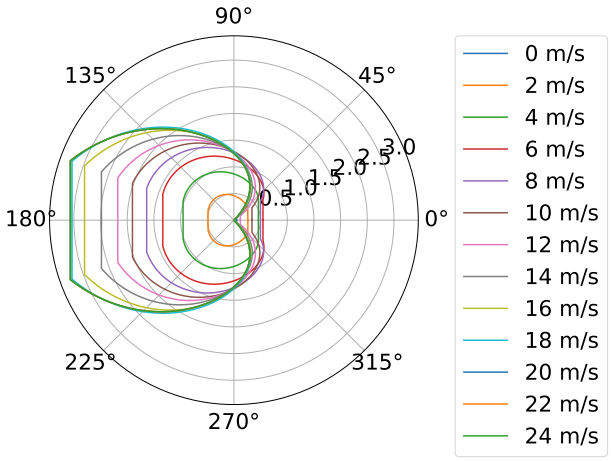


Fig. 2. Boat velocity polar in [m/s] wrt. point of sail [deg] for different wind magnitudes

C. Introducing stochastic boat performances

Until now the boat's dynamics were deterministic: a given wind and point of sail produce a unique boat velocity. This is not feasible since we did not fully model the boat or the sea state. It is known for example that wind gusts or waves drastically impact the boat's performances. To account for these physics we did not modelled, we add noise to the velocity of the speed polar. From the deterministic boat velocity $\mathcal{V}$ and the boat heading γ we can compute the boat velocity toward East U and toward North V as follow :

$$U = \mathcal{V}\sin\gamma, \qquad V = \mathcal{V}\cos\gamma \tag{1}$$

Then, we define a variance σ^2 proportional to the boat deterministic speed :

$$\sigma^2 = \kappa\mathcal{V} \tag{2}$$

where κ is an arbitrary constant set to 20%. Finally, the stochastic velocities u' and v' are picked from a normal distribution :

$$u' \sim \mathcal{N}(U, \sigma^2), \qquad v' \sim \mathcal{N}(V, \sigma^2) \tag{3}$$

D. Modelling the current

The VPP model considers static water surface, hence to get the actual velocity of the boat we must add the water surface velocity. The surface current can be approximated to be K=3% of the 10m AWSW [17]. Hence, for arbitrary wind velocities u_w and v_w, the total boat velocity is given by :

$$u = u' + Ku_w, \qquad v = v' + Kv_w \tag{4}$$

IV. The Interaction Model

In this section we describe how we built a simulator S that, for a given action (boat heading γ), and a given boat state $\boldsymbol{s}$, generates a new state $\boldsymbol{s'}$: $S(\boldsymbol{s}, \gamma) = \boldsymbol{s'}$.

A. The different blocks of the simulator

a) Boat states: a boat state is defined by a date t, a latitude φ and a longitude λ :

$$\boldsymbol{s} = (t, \varphi, \lambda)^{\mathrm{T}} \tag{5}$$

b) Wind conditions: the GFS and GEFS files are linearly interpolated over the domain. Hence, we get two functions $\boldsymbol{W}_{avg}(t, \varphi, \lambda)$ and $\boldsymbol{W}_{spr}(t, \varphi, \lambda)$ that respectively output the mean wind and the ensemble spread. Thus, for a given state $\boldsymbol{s}$, we can build the observed wind velocities u'_w and v'_w as follow :

$$\begin{aligned} \boldsymbol{W}_{avg}(\boldsymbol{s}) = (U_w, V_w)^{\mathrm{T}}, \qquad \boldsymbol{W}_{spr}(\boldsymbol{s}) = (\sigma_u, \sigma_v)^{\mathrm{T}} \\ (u'_w, v'_w) \sim (\mathcal{N}(U_w, \sigma_u^2), \mathcal{N}(V_w, \sigma_v^2))^{\mathrm{T}} \end{aligned} \tag{6}$$

c) Boat dynamic: From the last paragraph we get a stochastic wind condition for each given state. Coupling this wind condition with the boat and current model of section III for a given boat heading, we can compute a noisy boat velocity. We consider this velocity constant and we take its modulus and argument with respect to the true North. Then, given a time-step $\mathrm{d}t$, we get a covered distance $\Delta L = \sqrt{u^2 + v^2} \times \mathrm{d}t$ towards a direction ψ (that may differ from the boat heading due to the noisy wind and dynamic and the surface current), this step is thus the simplest integration possible.

d) Spherical geometry: Since the vessel is sailing on the spherical Earth surface we use the Geodesic formulae G_1 and G_2 to compute distances and headings. These formulae make the link between two points on the Earth surface $\boldsymbol{x_1} = (\varphi_1, \lambda_1)$, $\boldsymbol{x_2} = (\varphi_2, \lambda_2)$ and the distance D between them as well as the bearing θ from $\boldsymbol{x_1}$ to $\boldsymbol{x_2}$:

$$G_1(\boldsymbol{x_1}, \boldsymbol{x_2}) = (D, \theta)^{\mathrm{T}}, \qquad G_2(\boldsymbol{x_1}, (D, \theta)^{\mathrm{T}}) = \boldsymbol{x_2} \tag{7}$$

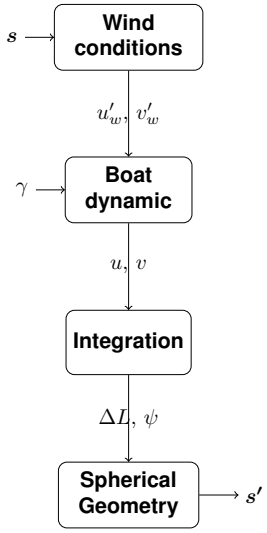


Fig. 3. Simulator's steps

B. Testing the Simulator

To test the simulator we launch 30 boats from the same departure state $\boldsymbol{s} = (0, 47.5, 356.5)^{\mathrm{T}}$ with a constant heading of 225° for a 6 days navigation. They all sail with the same GFS and GEFS data. To represent the trajectories we chose an Azimuthal Equidistant Projection centred on the departure state. This means that the distances from the departure are preserved and that all points that lie on a circle around the departure are equidistant on the surface of the Earth.

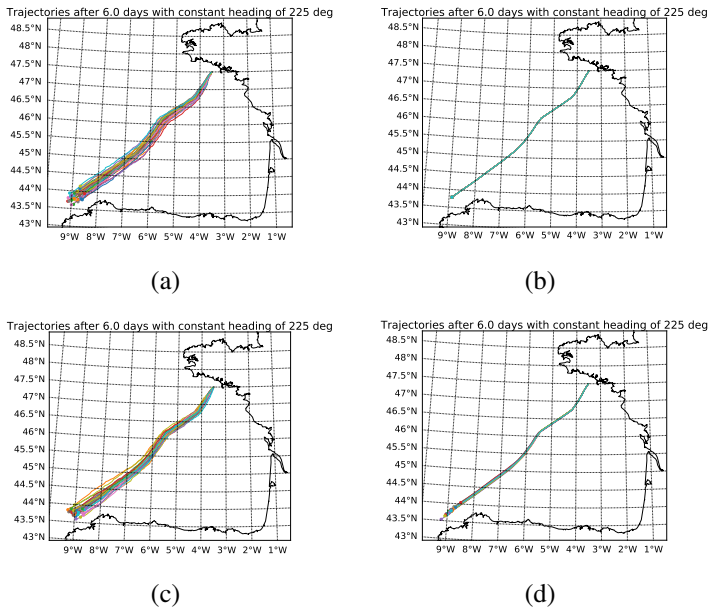


Fig. 4. 30 boats simulated with (a) stochastic wind and dynamics, (b) deterministic wind and dynamics, (c) deterministic wind and stochastic dynamics, (d) stochastic wind and deterministic dynamics

From figure 4 we can conclude that the simulator is coherent. Stochasticity has two effects: it spreads the boats trajectories and it modifies the distance between the arrival points and the departure. Without the stochastic modelling all trajectories are identical. It should be noted that the stochastic dynamic of the boat impacts more the spreading of the trajectory than the covered distance. The opposite occurs regarding the stochastic wind: the trajectories are close to one another but vary in length. This is due to the fact that the wind affects only the direction of motion of the boat through surface current (with a coefficient of 3%) but plays a major role in the velocity of the vessel. Whereas the boat's dynamic affect both its direction of motion and its speed.

Figure 5 displays a specific trajectory under stochastic wind and dynamics.

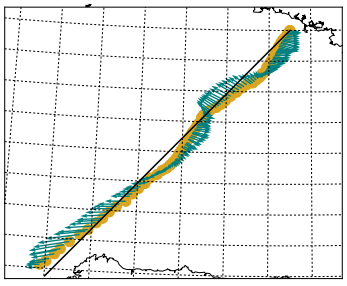


Fig. 5. One boat trajectory under uncertain wind and boat's dynamics

A yellow dot represents a state, the solid black line represents the constant 225° heading and the blue arrows are a quiver of the mean wind exerted on the boat at the corresponding state. This trajectory is coherent, we notice that the boat is deported away of the constant heading line by the wind. Finally the boat sails faster (dots are more spread out) when the wind come from behind (running point of sails).

V. The framework

A. Monte Carlo Tree Search (MCTS)

The definition of MCTS given by [3] is the following :

> Monte Carlo Tree Search is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results.

In other words, for large decision spaces, where an exhaustive search is not tractable, we pick random samples to explore the decision space. In order to estimate the value of the picked decision we run a simulation: the Default Policy. It can be a random or statistically biased sequence of actions applied to the state resulting from the picked decision. The Default Policy is applied until a terminal condition is reached. The terminal state that we reach is used to evaluate a reward which is incorporated in the value of the picked decision. A tree is built in an asymmetric and incremental manner in order to explore the decision space. Each node of the tree represents a sequence of decisions and possesses a value which is function of the reward obtained from the Default Policy.

B. Upper Confidence Bounds for Trees (UCT)

In order to build the tree in an asymmetrical manner, i.e. to explore only promising decisions and to avoid loosing time and computational power with inefficient decisions, we need a clever Tree Policy. A Tree Policy is a method that computes the values of the tree's nodes and that selects the most promising one. Thus, at each iteration of the search, the Tree Policy chooses the node to expand and thus dictates how the decision space is explored. In this work we will use the most popular algorithm in the MCTS family: Upper Confidence Bounds for Trees (UCT). We start at the root node of the tree and we select the child with the highest UCT value. If this child node is non-terminal and not fully expanded, we expand it. Otherwise, we look at the UCT values of its children and so on. The routine used to expand a node is described in section VI. The UCT value of a child node j is given by :

$$UCT = \overline{\Delta_j} + 2C_p\sqrt{\frac{2\log n}{n_j}} \tag{8}$$

Where $\overline{\Delta_j}$ is the mean reward of all the decisions sequences that passed through node j. n_j is the number of decisions sequences that involved the child node j, whereas n is the number of decisions sequences that involved the parent node. C_p is a positive coefficient: the exploration coefficient. The left-hand term represents the value we already obtained from node j. If it is high, then j is valuable and we should continue to exploit it. The right-hand side represents the exploration term. Indeed, if a child node is not explored, the right hand side increases as $\sqrt{\log n}$ with n the number of times the parent node has been explored. Thus, the UCT of unexplored nodes will eventually grow and have the Tree Policy select them.

One step of the MCTS-UCT algorithm is visually represented in figure 6.

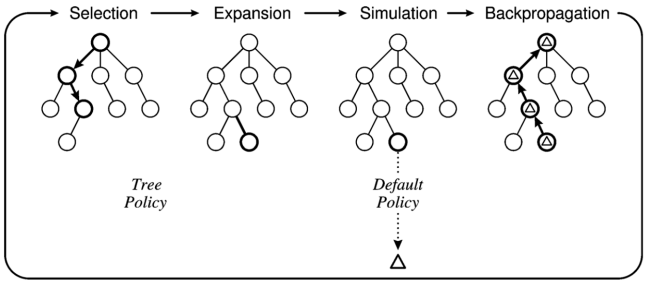


Fig. 6. One step of the MCTS-UCT algorithm generating a reward Δ, extracted from [3]

VI. The implementation of MCTS-UCT

A. Defining the objects

a) The available actions A: the purpose of the tree-search is to find the optimal sequence of actions, the prescription P, to go from a departure point $\boldsymbol{x_s}$ to a destination point $\boldsymbol{x_d}$. The optimal sequence is the one that minimizes, in average, the date of arrival. The actions are the headings followed by the vessel. Thus, they range from $0°$ to $360°$. In order to reduce the branching factor of the tree, the continuous range is reduced to a discrete set during the search. The set of headings starts at $0°$ and increases up to $360°$ with a step of $45°$.

b) A node ν: in this architecture only the root node ν_0 represents a state. The other nodes represent a sequence of actions executed from the initial state. The attributes of a node are:

- *state $\boldsymbol{s}$:* only for the root node. The state of the root node is the initial state $\boldsymbol{s_0}$ corresponding to $t = 0$ and the departure point $\boldsymbol{x_s}$.
- *parent:* a reference toward the parent node.
- *origin $a(\nu_l)$:* the action $a \in A$ taken from the parent to expand it and create the leaf node ν_l.
- *children:* a list of references toward the children nodes.
- *actions $\boldsymbol{a} \subset A$:* a shuffled list of the remaining available actions. Every time the node is expanded using one of the actions, this action is not available any more.
- *R:* sum of all the rewards Δ that passed through the node.
- *N:* number of times the node as been involved in a decision sequence.
- *depth:* corresponds to the number of actions taken from the root node to arrive to the node. Equivalent to the time increment from the initial state.

c) A tree: the tree is the object that collects the created nodes and possesses the methods implementing the MCTS search.

- *rootNode $\boldsymbol{s_0}$:* a reference pointing toward the root node object.
- *ite:* the number of nodes that have been created.
- *budget:* the total number of nodes that will be created during the search.
- *Simulator S:* a reference toward a simulator object like the one described in section IV.
- *destination $\boldsymbol{x_d}$:* the latitude and longitude of the destination provided by the initialization of the tree.
- *TimeMax T:* the temporal horizon on which the search is carried out. Also the horizon of the simulator.
- *TimeMin T_m:* an arbitrary estimate of the minimum time required by the boat to go from the departure to the destination. It is provided by the tree initialization.
- *depth:* maximum depth of the tree.
- *nodes:* a list of references toward the created nodes to keep track of the chronology of the tree's growth.
- *rewards:* a list of all the rewards that were obtained during the search.

B. Defining the processes

a) Initialization: this step converts a mission heading and a time horizon into a destination point and an estimated date of arrival. It can be divided in several sub-steps:

- *(1) Estimating a covered distance:* to estimate, given the wind conditions, the distance covered by the boat, $N_b = 50$ boats are simulated over the whole time horizon. They sail straight toward the mission heading.
- *(2) Computing the corresponding destination:* due to the drifting, the N_b boats do not actually sail with the correct heading. Therefore, the estimated covered distance is reduced taking a fraction c_d of it. The resulting distance is then projected toward the actual mission heading. The corresponding geographical point is the destination.
- *(3) Estimating a date of arrival:* then, to get an estimate of the date at which the boat will arrive to destination, N_b other boats are launched. These boats modify their heading to match the destination bearing, hence they eventually get to the destination. The fastest boat is taken as a reference.

Figure 7 shows the trajectories resulting from the initialisation of a search that has a time horizon of 2 days and a mission heading of $235°$. The blue point is the departure.

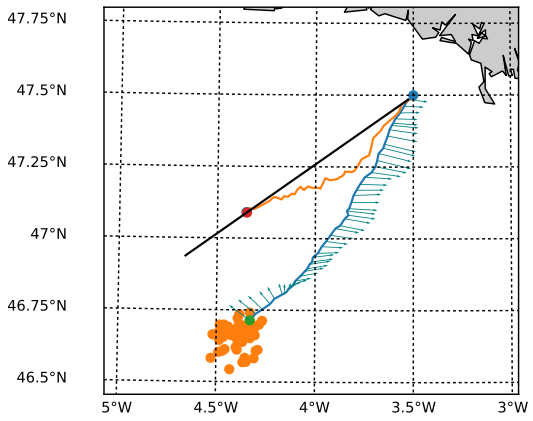


Fig. 7. Initializations trajectories

All the orange points are the arrivals of the N_b boats simulated in *(1)*. The blue trajectory is one of them. It can be observed that the boat drifted toward East due to the lateral wind. The red point is the destination obtained from *(2)*. The distance with respect to the departure is $c_d = 3/4$ of the minimal distance covered in *(1)*. The yellow trajectory is one of the boats sent to estimate the time of arrival in *(3)*. Finally the dark line indicates the mission heading.

b) The Default Policy: The Default Policy is to go straight toward the destination. A Default Policy followed from the departure would look like the yellow trajectory in figure 7. For an arbitrary node, it is necessary to first determine the corresponding state. This state can be estimated by starting from the root node and executing the sequence of actions corresponding to the node. Then, at each time step the destination bearing is computed to adapt the boat heading and reach the destination. Eventually, the reward is computed as :

$$\Delta = \exp\left(\frac{T \times c - T_d}{T \times c - T_m}\right) \tag{9}$$

Where T is the search time horizon and T_d is the date of arrival of the boat after following the Default Policy. T_m is obtained from step *(3)* of the initialization and c is a constant to avoid a division by zero. In our case $c = 1.001$. If the boat cannot reach the destination over the time horizon following the Default Policy the reward is set to zero. Thus, the reward is positive but can be greater than 1. An exponential reward instead of a linear one encourages boats that are faster than the initialisation ones. It also increases the distinction between promising solutions.

c) Terminal and fully expanded nodes: a state is terminal if it is at destination or if it reached the time horizon. Thus, a node is terminal if its depth corresponds to the time horizon. A node is fully expanded if it has no more actions available.

Algorithm 1: The UCT algorithm

function UCTSEARCH*(**s_0**)*:
  create root node ν_0 with state $\boldsymbol{s_0}$
  while within computational budget **do**
    ν_l, $\boldsymbol{s} \leftarrow$ TREEPOLICY$(\nu_0, \boldsymbol{s_0})$
    $\Delta \leftarrow$ DEFAULTPOLICY$(\nu_l, \boldsymbol{s})$
    BACKUP(ν_l, Δ)
  $\nu \leftarrow \nu_0$
  while ν has children **do**
    $\nu \leftarrow$BESTCHILD$(\nu, 0)$
    append $a(\nu)$ to P
return P

function TREEPOLICY*(ν, $\boldsymbol{s}$)*:
  while $\boldsymbol{s}$ is nonterminal **do**
    if ν not fully expanded **then**
      return EXPAND$(\nu, \boldsymbol{s})$
    else
      $\nu \leftarrow$ BESTCHILD(ν, Cp)
      $\boldsymbol{s} \leftarrow S(\boldsymbol{s}, a(\nu))$
return ν, $\boldsymbol{s}$

function EXPAND*(ν, $\boldsymbol{s}$)*:
  choose random $a \in \boldsymbol{a}(\nu)$ the untried actions of ν
  add a new child ν' to ν with $a(\nu') = a$
  $\boldsymbol{s} \leftarrow S(\boldsymbol{s}, a)$
return ν', $\boldsymbol{s}$

function BESTCHILD*(ν,C_p)*:
  $\nu_b \leftarrow \max\left(\frac{R(\nu')}{N(\nu')} + c\sqrt{\frac{2\log N(\nu)}{N(\nu')}}\right) \Big| \nu' \in$ children of ν
return ν_b

function DEFAULTPOLICY*(ν, $\boldsymbol{s}$)*:
  while $\boldsymbol{s}$ is nonterminal **do**
    D, $\theta \leftarrow G_1(\boldsymbol{x}, \boldsymbol{x_d})$ with $\boldsymbol{x}$ the position of $\boldsymbol{s}$
    $\boldsymbol{s} \leftarrow S(\boldsymbol{s}, \theta)$
return reward for state $\boldsymbol{s}$

function BACKUP*(ν,Δ)*:
  while ν is not null **do**
    $N(\nu) \leftarrow N(\nu) + 1$
    $R(\nu) \leftarrow R(\nu) + 1$
    $\nu \leftarrow$ parent of ν

This algorithm is adapted from Algorithm number 2 in [3].

VII. RESULTS

In this section the results of the search are presented. We propose explanations for what is observed. The limitations of the work are discussed. We suggest hints that could improve the model.

A. Discussion on the exploration coefficient

The exploration coefficient C_p in equation 8 is a critical parameter of UCT. As a matter of fact, it drives how much the tree search is greedy and looks for states that give "immediate" reward. For example, with a low exploration coefficient, nodes that in the beginning did not generate high rewards are abandoned and the tree quickly grows in the direction of the nodes that provided value. This is positive in the fact that, with such a logic, we focus on strategies that generate value and we do not loose time with the strategies that have low "short-term" reward. The danger is to miss an optimal solution that would require some initial investments (take some actions that do not result in a short arrival time using the Default Policy) but that ends up giving optimal results in the long run.

There is therefore a trade-off to find between reaching an important depth for a given computational budget and not missing the optimal solution. [3] advises a value of $1/\sqrt{2}$ but indicates that C_p is greatly problem dependent.

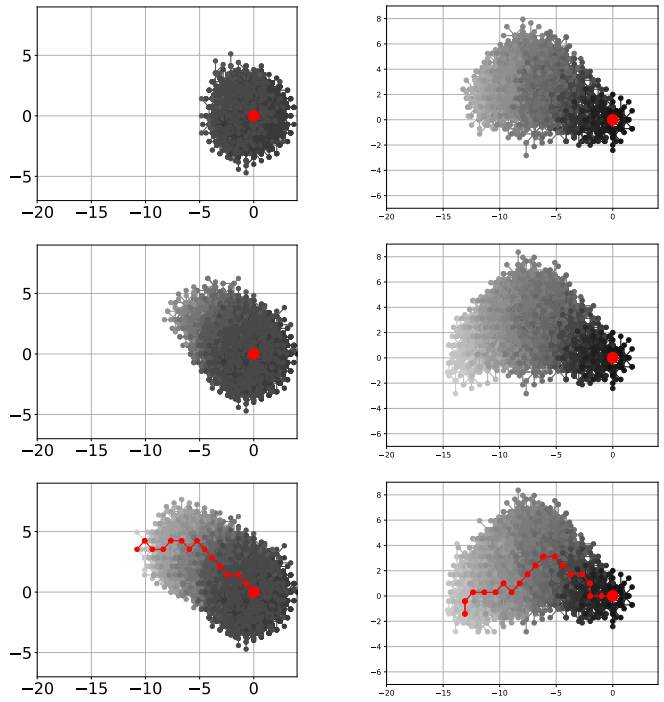
a) Asymmetrical tree growth: we show here the tree growth for two almost identical searches[5]. They have the same initialisation and the parameters of table II.

[5]The `Python` script takes 15 min on an Intel® Core™ i7-5500U CPU 2.40GHz

ite	$TimeMax$	time step	mission heading	c_d
15000	2 days	1 hour	235°	3/4

TABLE II
PARAMETERS OF THE SEARCH

They only differ in the exploration coefficient. Search (b) has a lower exploration coefficient and is therefore more "greedy".



(a) $C_p = 1/\sqrt{2}$, tree depth=14 (b) $C_p = 1/10$, tree depth=20

Fig. 8. Asymmetrical tree growth for two searches with different C_p

Each picture in figure 8 is a visual representation of the tree being built. Each dot is a node. The big red dot is the root node. The grey scale indicates the depth of the node. The lighter the color the deeper the node. Each node is placed around its parent depending on the action that expanded it. If the action from the parent node to the child node is for example 0° the child is place vertically above. Now if the action is 270°, the node is place horizontally on the left of its parent. The pictures are therefore polar representations of the tree. A column in figure 8 presents the chronological growth of the tree, with the early nodes on the top picture and the final tree on the bottom of the column. The red dotted line represents the optimal branch of the tree and corresponds to the best found sequence of actions. This sequence of actions is the output P of the UCT Search and we call it the prescription.

It can be observed that the trees differ in depth, in shape and in their prescription. The tree (a) is denser but tree (b) is deeper. This means that (a) tried more alternatives. It starts by exploring symmetrically around the root node. Once it has found a preferred direction (around 315°) it starts exploring preferentially in this direction but keeps expanding a few nodes around the root. On the other hand, (b) focuses only in one direction from the root and this choice is never questioned.

The prescribed sequence of actions P are :

$P_{(a)}$: $(315, 315, 270, 315, 315, 315, 315, 225, 315, 270, 225, 270, 315, 225)^{\mathrm{T}}$

$P_{(b)}$: $(270, 270, 0, 315, 270, 315, 315, 270, 225, 225, 225, 225, 315, 225, 270, 270, 225, 180, 0)^{\mathrm{T}}$

B. Consequences on the arrival date

In this subsection we test the prescriptions.

a) The test procedure: for each sequence of actions, 2400 boats are launched. Each boat follows N_a actions from the prescription (either the whole sequence or a subset) and then follows the Default Policy towards the destination. The Default Policy (going in a straight line toward objective) can be done either with continuous headings (Continuous) or with headings from the discrete set (Discrete). Of course a Continuous Default Policy is more effective since the steering is more precise. Also, Default Policy applied from the departure point is tested, this is what novice navigator do in practice. The arrival dates are measured in time steps (hours), the whole navigation horizon representing 48 time steps of one hour (2 days).

b) The mean dates of arrival: The results of these tests are consigned in table III. This last one is sorted : best strategy is first.

Test n°	P_i	N_a	**Default Policy**	**Mean Arrival Date**
1	(a)	10	Discrete	45.117
2	(a)	10	Continuous	45.424
3	(a)	14	Discrete	45.753
4	-	0	Continuous	45.900
5	(b)	14	Discrete	45.936
6	(b)	14	Continuous	46.058
7	-	0	Discrete	46.625
8	(b)	20	Discrete	46.787

TABLE III
RESULT OF THE TESTS ON THE DIFFERENT STRATEGIES

First of all the (a) strategy is better than the (b) one in all cases. The C_p coefficient, that characterizes the exploration-exploitation trade off is thus more pertinent in the (a) search. In the (b) search, C_p is set so low that too little exploration takes place. Hence, the search is confined in a sub-optimal solution without being challenged by the exploration of alternatives. Moreover, with a low exploration coefficient, the (b) search is more sensible to the noise present in the reward model. Eventually, the tree reaches a significant depth but misses the good solution. We can conclude that it is more effective to have a reduced but optimal sequence of actions than a significant but sub-optimal prescription.

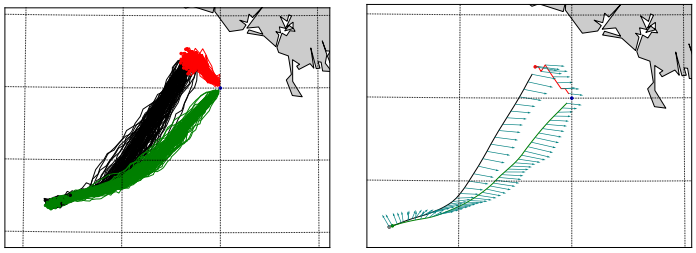
Secondly, to get the best out of a prescription, it should not be used to its final depths. Indeed, using only the 10 first actions of (a) is more profitable than using all of them. This is due to the fact that the last 4 recommended actions have not been tried much by the UCT algorithm. Therefore, the actual value of these decisions is not accurately predicted by

the mean of the obtained rewards. This might explain why it is more profitable to run only the first ten steps of $P_{(a)}$.

Thirdly, a prescription should be used with the Default Policy that has been used during the search. This may seem trivial since the strategy has been optimized for this particular Default Policy. However, it is not trivial that the performance gain from the optimal strategy outpaces the performance loss from using discrete headings during the Default Policy. This performance loss is significant as we can see in table III. Indeed, using the Default Policy from the departure is faster by 43 min if the heading are Continuous.

Finally, the main result is that the sequence derived from search (a), Test n°1, is faster by 43 min than the novice, but continuous steering Test n°4.

c) The trajectories: To get a better understanding of the strategy produced by the search, figure 9 presents the trajectories corresponding to table III.



(1) 100 test trajectories

(2) mean test trajectories and corresponding wind

Fig. 9. Test trajectories under prescription and Discrete Default Policy (red and black) and under Continuous Default Policy (green)

The red and black trajectories correspond to Test n°1 in table III, the red part corresponds to the N_a actions of the prescription and the black trajectories are the corresponding Default Policies. The green trajectories represent Test n°4 and are generated by a Continuous Default Policy applied from the departure. One can notice that the optimal route uses a tacking route. This is really interesting because the model does not explicitly take into consideration the tacking action. However, the search is capable of rediscovering this well-known strategy to compensate for the low velocity of the vessel when it is sailing into the wind. By following this tacking route, the boat covers more distance but has better points of sail along its course. Eventually it is faster over the whole trajectory and wins the race.

For the comparison, the green trajectory has a more direct route, however, it sails closer to the wind and has thus a reduced velocity over the whole race.

C. Perspectives

We have seen that Monte Carlo Tree Search could be efficiently applied to long term path planning for off-shore sailing. This problem is highly uncertain and combines stochastic wind forecasts, whose accuracy decay with time, and noisy boat performances, that affect its velocity both in magnitude and direction. However, the algorithm is able to retrieve a navigation strategy employed by sailors. In addition, following a prescription during only the first quarter of the cruise is sufficient to gain 43 minutes on a two day journey.

The work done here could be generalized to continuous sets of actions using Double Progressive Widening as it has been proposed by [5]. However, having a heading resolution of 45° is not that restrictive. Indeed, the aim of the path planning, or high level control, is to give a general direction of motion to the system. Lower level controls operate then around the prescribed heading to optimize short term performances from instantaneous and noisy measurements.

REFERENCES

[1] Lamia Belouaer, Mathieu Boussard, and Patrick Bot. Strategic decision making in yacht match racing: Stochastic game approach. In *The 5th High Performance Yacht Design Conference*, pages 150–159, 2015.

[2] Y. Briere. Iboat: An autonomous robot for long-term offshore operation. In *MELECON 2008 - The 14th IEEE Mediterranean Electrotechnical Conference*, pages 323–329, May 2008.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[4] Guillaume M. J-B. Chaslot, Mark H. M. Winands, H. Japp Van Dan Herik, Jos W. H. M. Uiterkijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 04(03):343–357, 2008.

[5] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. *Continuous Upper Confidence Trees*, pages 433–445. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6] Daniel S. Ferguson and Pantelis Elinas. *A Markov Decision Process Model for Strategic Decision Making in Sailboat Racing*, pages 110–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[7] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 273–280, New York, NY, USA, 2007. ACM.

[8] George L. Hanssen and Richard W. James. Optimum ship routing. *Journal of Navigation*, 13(3):253–272, 1960.

[9] A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis lectures on artificial intelligence and machine learning. Morgan & Claypool, 2012.

[10] David Martin and Robert F Beck. Pcsail, a velocity prediction program for a home computer. In *15th Chesapeake Sailing Yacht Symposium*, volume 100, 2001.

[11] Andy Philpott and Andrew Mason. Optimising yacht routes under uncertainty. In *The 15th Cheasapeake Sailing Yacht Symposium*, 2001.

[12] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[13] C. Pêtrès, M. A. Romero-Ramirez, and F. Plumet. Reactive path planning for autonomous sailboat. In *2011 15th International Conference on Advanced Robotics (ICAR)*, pages 112–117, June 2011.

[14] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[15] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172. Curran Associates, Inc., 2010.

[16] Roland Stelzer and Tobias Pröll. Autonomous sailboat navigation for short course racing. *Robotics and Autonomous Systems*, 56(7):604 – 614, 2008.

[17] Jan Erik Weber. Steady wind- and wave-induced currents in the open ocean. *Journal of Physical Oceanography*, 13(3):524–530, 1983.

[18] Yuejian Zhu and Zoltan Toth. Ensemble based probabilistic forecast verification. 2008.