Exploring Routing Strategies for a Virtual Yacht Race

90 days around the world through the LiveSkipper experience

Yves A. Papegay

Odile Pourtallier

INRIA - Sophia Antipolis Méditerrannée
yves.papegay@sophia.inria.fr
odile.pourtallier@sophia.inria.fr

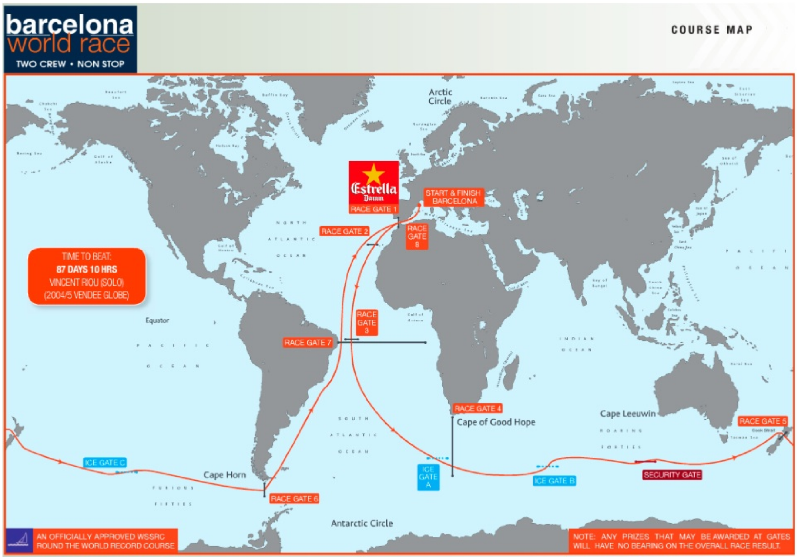
In this talk, we describe how we used *Mathematica* to implement an around the world racer, automatically connecting the LiveSkipper web site five times an hour during the whole race, getting informations on its own boat, computing new heading and sail configuration corresponding to the pre-defined routing strategy and taking in account the geographical constraints, and sending this control back to the web site.

```
<< "Desktop/Liveskipper.m"
<< Geodesy`
<< Units`
```

The Barcelona World Race

The Barcelona World Race is a two-handed, non-stop round the world sailing yacht race started from Barcelona on the 11th November, 2007. Participants are racing more than 28,000 miles over three months across the planet's most hostile and challenging oceans in high-performance IMOCA Open 60 monohulls.

■ The Barcelona World Race (cont'd)



LiveSkipper

The LiveSkipper web site offers an opportunity to virtual skippers to compete against the real skippers in this sailing competition. Thanks to the LiveSkipper simulation software, it is possible to participate in the same conditions i.e. with the same boat performances and the same weather changes as the real champions.

Interface

Simulation

The LiveSkipper simulation engine hosted by the web site updates every 10 minutes the position of each boat, based on its previous position and on its computed speed.

Global Earth Model

- Location of the seas end of the coasts

Weather Model

- Wind direction and speed comes from real meteorological data observed and forecasted, given as a 0.5 degree by 0.5 degree grid updated every three hours.

Boat Model

- The speed of each virtual ship is computed as a function of the following parameters :

  a. the wind direction and speed at her location

  b. her heading,

  c. her sail configuration.

Getting/Setting Boat Parameters

Challenge consists in controlling the heading and the sail configuration often enough to maximise the speed of the boat, with respect to the local wind conditions together with a global routing strategy. The success, of course, also depends on the quality of the routing strategy.

Provided that we can connect automatically the LiveSkipper interface, we can save several sleeping hours by letting *Mathematica* be our automatic pilot : it should be of great help to control the boat parameters, to analyse the meteorogical and geographical conditions and to compute the best sailing route around the world.

through the Graphical Interface

It is possible to connect autmatically the interface as an human player would do, i.e. by launching a web browser, navigating through the site and through the identification process, getting the flash interface on the screen, taking a snapshot of the corresponding window, analysing this image to find the boat information, computing the new parameters using a pre-defined strategy, and then mimicking some mouse events to set these new parameters.

In such a complicated process, Mathematica is of great help for the image analysis and the computation of the parameters. Other operation rely on operating system tools - Automator, Capture in the Mac OSX case - and specialized web automation tools designed for user interface testing - we used iMacros Firefox plug - in

Snapshot Analysis

```
img = Import["Desktop/liveskipper/liveskipper.tiff"]
```

Representation of the Image

```
img[[0]]
Length[img]
img[[1]] // Short
img[[2]]
```

Graphics

3

```
Raster[{{{255, 255, 255, 255}, {255, 255, 255, 255}, «805»,
    {255, 255, 255, 255}, {255, 255, 255, 255}}, «568»},
  {«1», «1»}, «1», ColorFunction → RGBColor]
```

ImageSize → {809, 569}

Disgrading of the Image

```
Levelize[n_Integer, p_Integer] := p Quotient[n, p]
```

```
Map[Levelize[#, 8] &, Range[32] - 1]
```

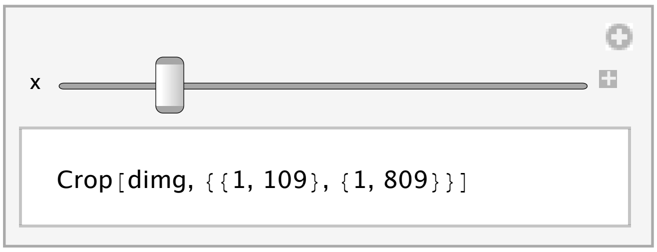
```
{0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 16, 16,
  16, 16, 16, 16, 16, 16, 24, 24, 24, 24, 24, 24, 24, 24}
```

```
dimg = img /. {r_Integer, g_Integer, b_Integer, a_Integer} →
    Map[Levelize[#, 32] &, {r, g, b}]
```

□ Select the Information Area

```
Crop[img_, {{lm_, lM_}, {cm_, cM_}}] :=
  ReplacePart[MapAt[#[[lm ;; lM, cm ;; cM]] &, img, {1, 1}],
    {{1, 2, 2} → {cM - cm, lM - lm}, {2, 2} → {cM - cm, lM - lm},
      {3, 2, 1, 2} → cM - cm, {3, 2, 2, 2} → lM - lm}]
```

```
Manipulate[Crop[dimg, {{1, x}, {1, 809}}], {x, 9, 569, 20}]
```



```
simg = {Crop[dimg, {{13, 27}, {99, 110}}],
    Crop[dimg, {{13, 27}, {110, 121}}],
    Crop[dimg, {{13, 27}, {127, 138}}]}
```

□ Analyze Information

```
(Map[MatrixForm,
    (nd = Map[(#[[1, 1]] /. x_Integer → If[x > 128, 1, 0] /.
            {r_, g_, b_} → r + g + b /. x_Integer →
            If[x > 1, 1, 0]) &, simg])]) /. {1 → "B", 0 → "."}
```

```
{ . . . . . .
. . B B B B B B . . .
. . . . . .
. . . . . B . . .
. . . . . B . . .
. . . . . B . . .
. . B B B B B B . . .
. . . . . .
. . . . . B . . .
. . . . . B . . .
. . . . . B . . .
. . B B B B B . . . .
. . . . . .
. . . . . .
. . . . . ,

. . . . . .
. . . B B B B B B . . .
. . . . . B .
. . . . . B B .
. . . . . B B .
. . . . . B B .
. . . B B B B B B . . .
. . . . . .
. . B . . . . . B B .
. . B . . . . . B B .
. . B . . . . . B B .
. . . . B B B B B . . .
. . . . . .
. . . . . .
. . . . . ,

. . . . . .
. . . . . .
. . . . . B . .
. . . . . B . .
. . . . . B . .
. . . . . B . .
. . . . . .
. . . . . B . .
. . . . . B . .
. . . . . B . .
. . . B B B B B . . . .
. . . . . .
. . . . . .
. . . . . }
```

```
f[{{z__}, x__}] /; z == 0 := {x}
f[{x__, {z__}}] /; z == 0 := {x}
f[x_] := x
```

```
(Map[MatrixForm,
    (nds = Map[Transpose[FixedPoint[f, Transpose[FixedPoint[
            f, #]]]] &, nd])]) /. {1 → "B", 0 → "."}
```

```
{ B B B B B B .
  . . . . . . .
  . . . . . . B
  . . . . . . B
  . . . . . . B
  B B B B B B .
  . . . . . . .
  . . . . . . B
  . . . . . . B
  . . . . . . B
  B B B B B . . ,

  . B B B B B B . .      . . . . . . B
  . . . . . . . . B      . . . . . . B
  . . . . . . . B B      . . . . . . B
  . . . . . . . B B      . . . . . . B
  . . . . . . . B B      . . . . . . .
  . B B B B B B . .  ,   . . . . . . .
  . . . . . . . . .      . . . . . . B
  B . . . . . . B B      . . . . . . B
  B . . . . . . B B      . . . . . . B
  B . . . . . . B B      B B B B B . . }
  . . B B B B B . .
```

```
GetNumber[simg]
```

397

Getting/Setting Boat Parameters through PHP

The PHP access to the LiveSkipper interface provides the following URLs to get and set the boat information. Prior authentication using cookies is required.

---

http://www.liveskipper.com/GameEngine/flash/getBoatInfos.php?raceId=$id
http://www.liveskipper.com/GameEngine/flash/getraceinfos.php?raceId=$id
http://www.liveskipper.com/GameEngine/flash/sendBoatInfos.php?raceId=$id
&sailconfig=$sail&heading=$heading

The **Import** function allows to import data from any accessible URL, using whatever proxy and related settings are specified in the user environment, but unfortunately this function do not consider cookies or other authentication.

To overcome this restriction, we developped two tiny perlscripts that realize the connection to the web site, the authentication and the exchange of information. The *Mathematica* commands **GetBoatInfos[]** (resp. **SetBoatInfos[heading, index]**) run the corresponding perlscript, exchanging information through a stream and a pipe. Namely, **GetBoatInfos[]** returns the following formated information on the boat and its environment: ranking, time stamp, coordinates (latitude, longitude in degrees) wind speed at the boat location, wind direction at the boat location, heading of the compass.

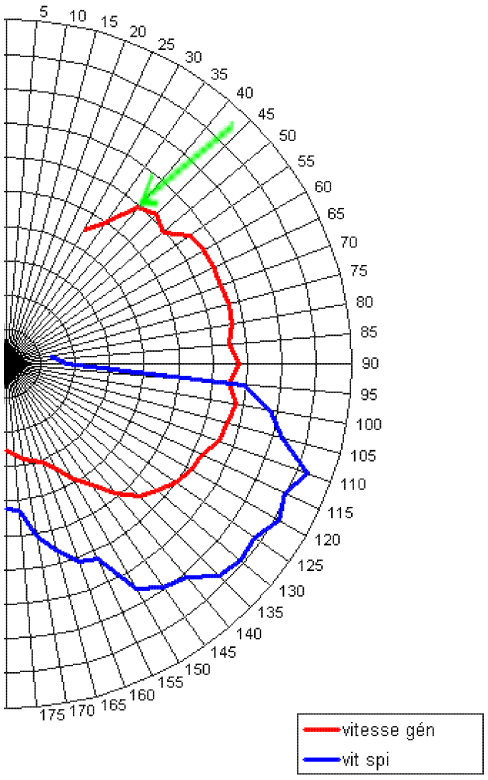
```
GetBoatInfos[] := With[{s = OpenRead["! perl
                    Desktop/liveskipper/perlscript/
                    GetBoatInfos.pl"]},
  LogBoatInfos[Table[Read[s, String], {8}]]]
```

```
GetBoatInfos[]
```

```
{616, TimeStamp[2008, 2, 9, 15, 28],
 Coord[{36, 12, 50}, {0, -9, -32}], 16.6, 55, 100}
```

About Boat Speed

Polar Curves

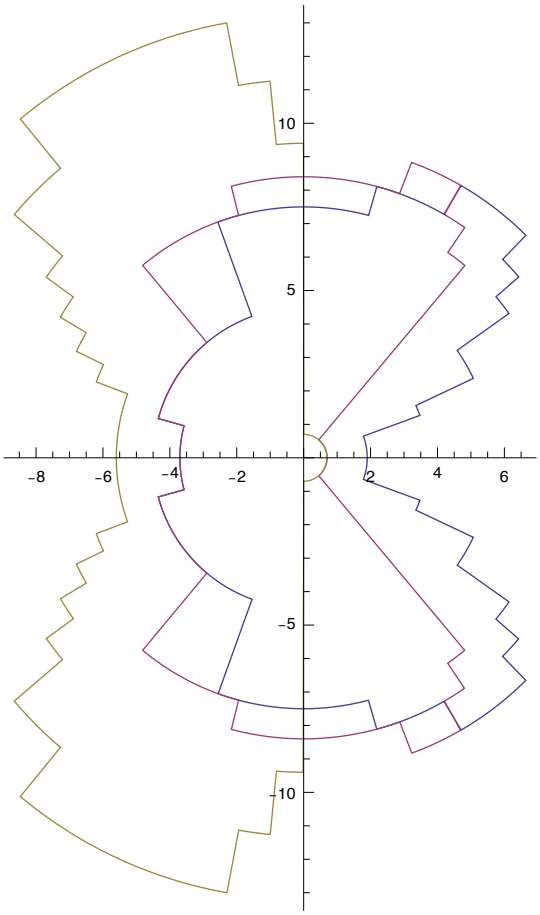


Vitesse du vent : 13Kn.

```
BoatSpeed[ws_, wd_, r_, i_] := BoatSpeed[ws, r - wd, i]

BoatSpeed[w_, rr_, i_] /; x < 0 := BoatSpeed[w, -rr, i]
BoatSpeed[w_, rr_, i_] /; x > 180 := BoatSpeed[w, rr - 360, i]
```

```
PolarPlot[
  {BoatSpeed[10, 180 x/Pi, 2], BoatSpeed[10, 180 x/Pi, 3],
    BoatSpeed[10, 180 x/Pi, 4]}, {x, -Pi, Pi}]
```



About Navigation

For navigation, position is given as Coord[latitude, longitude] where both angles can be decimal_degree, {degree, minute}, or {degree, minute, second}. By convention, north latitude and east longitude are positive. Except for zero, the sign of degree component is the one taken ito account.

For geography, position is given as {longitude, latitude} using decimal degrees.

```
ToXY[Coord[x1_List, x2_List]] := Map[ToDegrees, {x2, x1}] // N
FromXY[{long_, lat_}] := Apply[Coord, Map[ToDMS, {lat, long}]]

FromXY[{-119.406, -56.384}]
ToXY[%]
```

```
Coord[{-56, -23, -2}, {-119, -24, -22}]
```

```
{-119.406, -56.3839}
```

Distance

```
Distance[x1_Coord, x2_Coord] :=
  SphericalDistance[x1 /. Coord → List, x2 /. Coord → List]
    Convert[Kilo Meter, NauticalMile]
```

There are several ways to represent the earth on a planar chart. Most nautical charts use Mercator projection which is an orthogonal projection on a cylindrical tangent to the earth at equator. Paralleles are represented by horizontal lines, while meridians are represented by lines orthogonal to the parallels. The most important feature of this representation is that it preserves angles, which are the information availables (heading, wind and current directions ...) when cruising. Nevertheless the relative distances (and hence shapes) are not preserved with mercator projection. Distances are stretched when mouving away from equator.

$$\frac{2\pi}{360} R_{\text{earth}} \cos^{-1}(\cos(L_1)\cos(L_2)\cos(l_2 - l_1) + \sin(L_1)\sin(L_2)) \quad (1)$$

with L = latitude
and l = longitude

Route

An astonishing consequence of that is that the shortest route from one point O to an other point D (called orthodromic route), is not in

general represented by a straight line on the map, whereas a straight line route on the chart from O to D (called loxodromic route), that is route with constant heading, is not the shortest route.

```
Route[x1_Coord, x2_Coord] :=
  Round[With[{a1 = Map[ 2 Pi ToDegrees[#] / 360. &, x1],
      a2 = Map[2 Pi ToDegrees[#] / 360. &, x2]},
    With[{c1 = ArcCos[Sin[Latitude[a1]] Sin[Latitude[a2]] +
          Cos[Latitude[a1]] Cos[Latitude[a2]]
            Cos[Longitude[a1] - Longitude[a2]]]},
      With[{r = 180./Pi Re[ArcCos[(Cos[Pi/2 - Latitude[a2]] -
                Cos[Pi/2 - Latitude[a1]] Cos[c1]) /
              (Sin[Pi/2 - Latitude[a1]] Sin[c1])]]},
        If[Longitude[a1] < Longitude[a2], r, 360 - r]]]]]
```

About Geography

```
NorthernMost[l_] := FromXY[First[Sort[l, #1[[2]] > #2[[2]] &]]]
SouthernMost[l_] := FromXY[Last[Sort[l, #1[[2]] > #2[[2]] &]]]
EasternMost[l_] := FromXY[First[Sort[l, #1[[1]] > #2[[1]] &]]]
WesternMost[l_] := FromXY[Last[Sort[l, #1[[1]] > #2[[1]] &]]]
```

Canarias

```
Manipulate[Graphics[
  {Polygon[Join[CountryData["Morocco", "FullPolygon"][[1]],
      CountryData["Spain", "FullPolygon"][[1]]]], Red,
    Polygon[CountryData["Spain", "FullPolygon"][[1, i]]]}], {i,
  Range[Length[CountryData["Spain", "FullPolygon"][[1]]]]}]
```

```
NorthernMost[CountryData["Spain", "FullPolygon"][[1, 13]]]
```

```
Coord[{29, 25, 35}, {-13, -32, -43}]
```

Short Scale Routing

(very) local optimization

```
RouteBoatSpeed[ws_, wd_, th_, r_, i_] :=
  BoatSpeed[ws, wd, r, i] Cos[(r - th) Pi/180]

BestRoute[ws_, wd_, th_, i_] :=
  With[{t = Table[RouteBoatSpeed[ws, wd, th, y, i],
        {y, th - 90, th + 90}]}, With[{m = Max[t]},
    With[{br = r - 90 + Position[t, m][[1, 1]] - 1},
     {m, br, BoatSpeed[ws, wd, br, i]}]]]

BestRoute[ws_, wd_, th_] := Apply[{#[[1, 2]], #[[2]]} &,
   Sort[Map[{BestRoute[ws, wd, th, #], #} &, {2, 3}],
    #1[[1, 1]] > #2[[1, 1]] &]]
```

automatic routing

```
AutoPilot[tgt_, i_] := SetBoatInfos[With[{bi = GetBoatInfos[]},
     BestRoute[bi[[4]], bi[[5]], tgt, i]][[2]], i]
AutoPilot[tgt_Coord, i_] := SetBoatInfos[
   With[{bi = GetBoatInfos[]}, BestRoute[bi[[4]],
     bi[[5]], Route[bi[[3]], tgt], i]][[2]], i]

AutoPilot1[tgt_] := Apply[SetBoatInfos, With[{bi = GetBoatInfos[]},
     BestRoute[bi[[4]], bi[[5]], Route[bi[[3]], tgt]]]]
```

Short scale routing

Our goal is to determine the minimal time route to go from one origine point O to some destination point D.

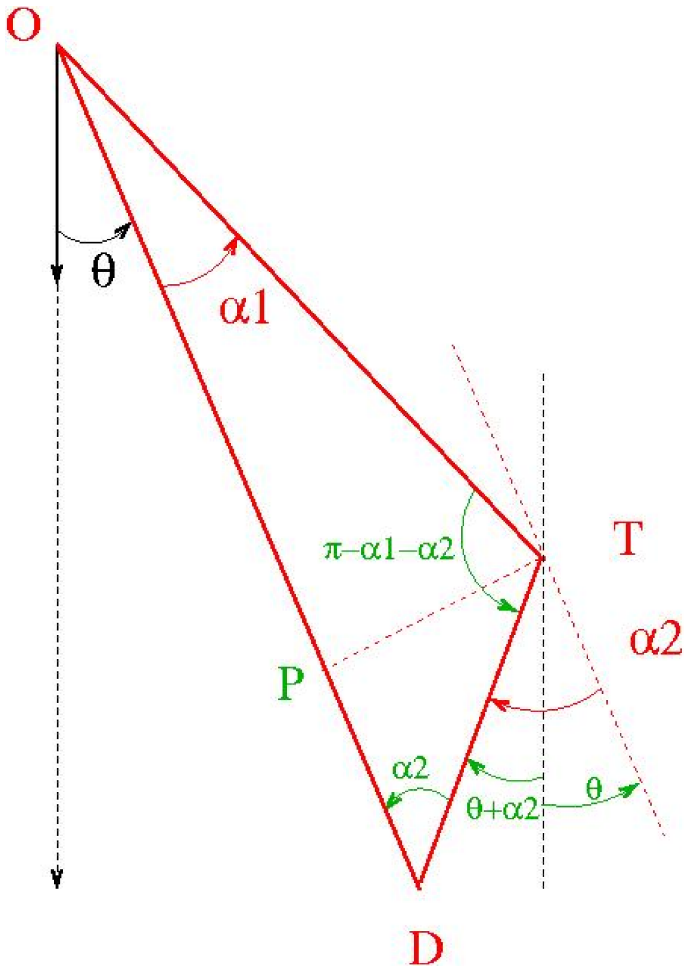
By short scale routing we mean that points O and D are such that there exists an geographical area containing O and D, such that

- the roundness of earth can be neglected. In particular loxodromic route is a good approximation of orthodromic route joining any two points of this geographical area,
- the wind is constant both in intensity (denoted w) and direction and will remain constant during the whole trip !

Denote θ the angular between the wind direction and the direct route OD. The time necessary to go directly from O to D is hence given by

$$t = \frac{\mathrm{OD}}{v_w(\theta)} \tag{2}$$

Let us now consider an alternative non direct route that consists in sailing first in a direction OT during a time t_1, and then in a second direction TD during time t_2.



This route is admissible if it ends to point D, that implies

$$t_1\, v_w(\theta+\alpha_1)\sin(\alpha_1) = t_2\, v_w(\theta+\alpha_2)\sin(\alpha_2) \tag{3}$$

$$t_1\, v_w(\theta+\alpha_1)\cos(\alpha_1) + t_2\, v_w(\theta-\alpha_2)\cos(\alpha_2) = \mathrm{OD} \tag{4}$$

```
equation1 =  t1  v1  Sin[a1] == t2  v2  Sin[-a2]
```

t1 v1 Sin[a1] == -t2 v2 Sin[a2]

```
equation2 = t1  v1  Cos[a1] + t2 v2 Cos[a2] == OD
```

t1 v1 Cos[a1] + t2 v2 Cos[a2] == OD

```
t = t1 + t2 /. First@Solve[{equation1, equation2}, {t1, t2}]
```

$$\frac{\text{OD Sin}[a1]}{v2\,(\text{Cos}[a2]\,\text{Sin}[a1] - \text{Cos}[a1]\,\text{Sin}[a2])} - \frac{\text{OD Sin}[a2]}{v1\,(\text{Cos}[a2]\,\text{Sin}[a1] - \text{Cos}[a1]\,\text{Sin}[a2])}$$

From this it comes that the total duration for the two leg route is

```
Factor@TrigReduce[t]
```

$$\frac{\text{OD Csc}[a1 - a2]\,(v1\,\text{Sin}[a1] - v2\,\text{Sin}[a2])}{v1\,v2}$$

The minimal value of the time t depends upon the expression of the function giving the speed of the boat and can only be approximated numericaly.

```
MaxBoatSpeed[ws_, wd_, r_] :=
  Max[Map[BoatSpeed[ws, wd, r, #] &, {2, 3, 4}]]
```

```
Off[General::infy]
LocalBestRoute[ws_, wd_, th_] :=
  With[{t = Table[With[{v1 = MaxBoatSpeed[ws, wd, r1 + th],
              v2 = MaxBoatSpeed[ws, wd, -r2 + th],
              a1 = (Pi/180) r1, a2 = (Pi/180) (-r2)},
            Csc[a1 - a2] (v1 Sin[a1] - v2 Sin[a2]) / (v1 v2)],
          {r1, 1, 89}, {r2, 1, 89}]},
    With[{m = Min[t /. ComplexInfinity → 100]},
      With[{p = First[Position[t, m]]},
        {p[[1]] + th, -p[[2]] + th}]]]
```

```
LocalBestRoute[10, 30, 130]
```

{131, 86}

Long Scale Routing a.k.a. Perspectives

Whenever the point O and D are not closed, the previous analysis cannot be used anymore since the wind is not likely to be constant and since larger scale the roundness of earth cannot be neglected anymore.

The isochrone method based upon forward induction is popular in this cases in routing softwares.

The underlying idea of the algorithm is to determine, by induction, the reachable points at each step, starting at initial time at origine point O.
As a matter fact, if Ω_k is the set of points that can be reachable in at most k steps, then starting from each point P of the boundary of Ω_k, given the wind direction and intensity, one can determine the set of points that are reachable in one step from P, and hence the set of points that are reachable in $k+1$ steps. Note that for each points of Omega_ 1 (P) one can keep track of the point P. The algorithm stop when destination point D is contained in the reachable set.