

Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

PLAN

1st session

Introduction (15-30 minutes)

C++ (1 hour)

The Clarke and Wright algorithm

Modifying the CAW and testing it

Adding your own strategy to the CAW

How to debug ?

2nd session

JSON (30 minutes)

Understanding the role of the JSON files

Adding two new algorithms to the CAW

PHP (30 minutes)

Adding a new problem to the web-site

Conclusion will be yours !



Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

C++

The Clarke and Wright algorithm.

Modifying the CAW and testing it.

Adding your own strategy to the CAW.

How to debug ?



Clarke and Wright, C++

A nice demo ☺ to start !

5 GPS points in Paris

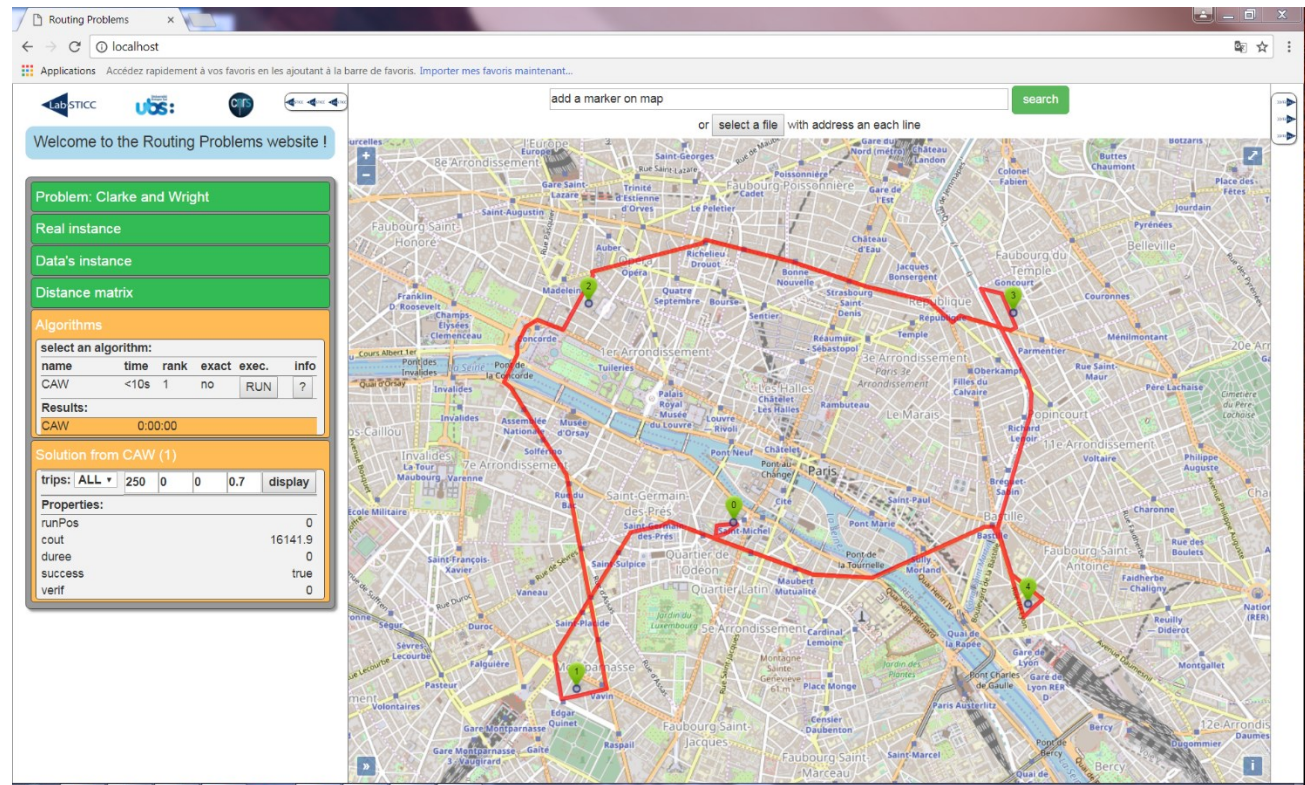
0 = pizza shop

1..4 are the customers

CAW will compute a list of paths to deliver the pizzas to the customers, minimizing the total distance traveled by the vehicle.

The vehicle has a maximum capacity and may need several paths to deliver all the pizzas to all the customers.

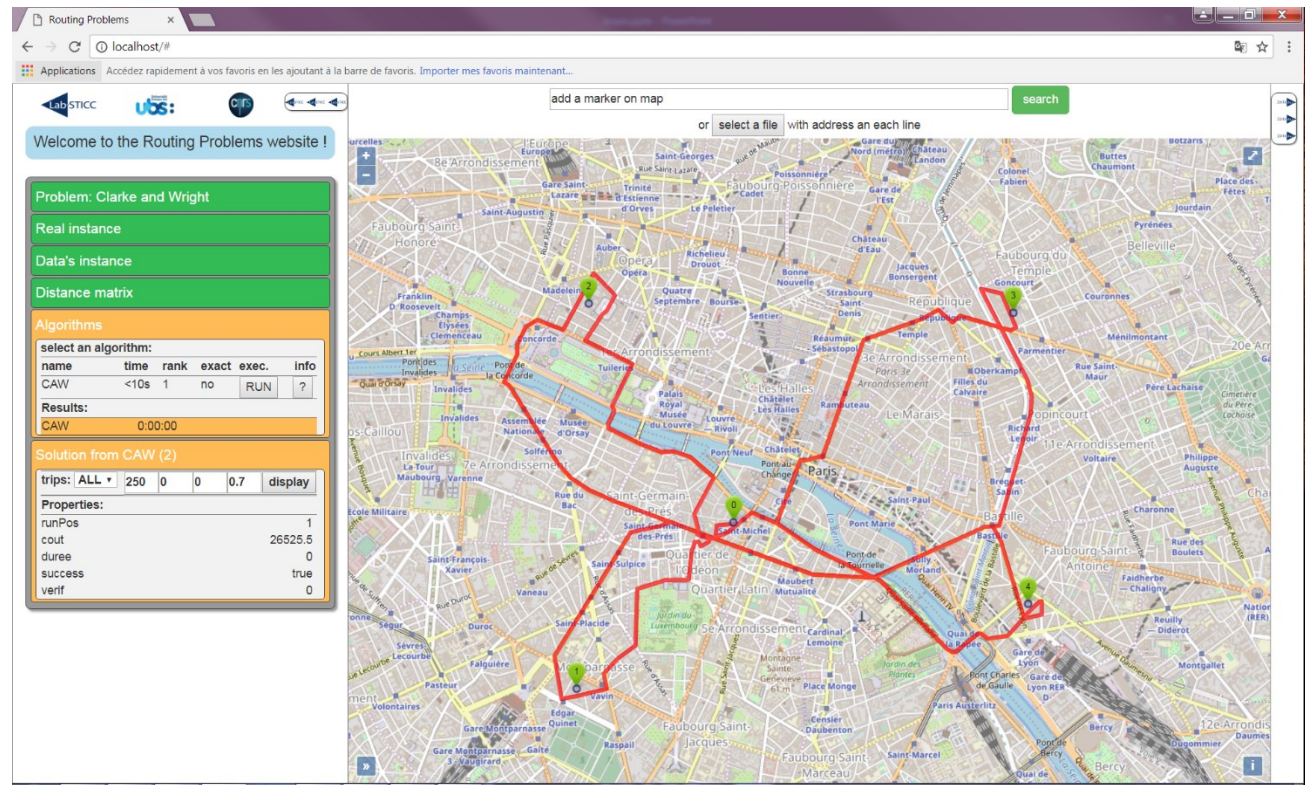
Here is a example with a single path.



Clarke and Wright, C++

The algorithm first builds the set of all the paths from the depot (pizza shop) to all the customers.

Because we have 4 customers we can see here 4 paths.



Clarke and Wright, C++

Second, the algorithm chooses the best pair of paths to merge into a single one.

It computes all the possible mergings and takes the best.

The best is the merge that saves the maximum distance.

Here we can see the result (3 paths) after the first iteration.

The screenshot shows the 'Routing Problems' website interface. The left sidebar contains a navigation menu with the following items: 'Problem: Clarke and Wright', 'Real instance', 'Data's instance', 'Distance matrix', 'Algorithms', 'select an algorithm:', 'Results:', and 'Solution from CAW (3)'. The 'Algorithms' section displays a table with columns: name, time, rank, exact, exec., and info. The 'Results' section shows the CAW algorithm's performance. The 'Solution from CAW (3)' section displays the results of the first iteration, showing 3 paths.

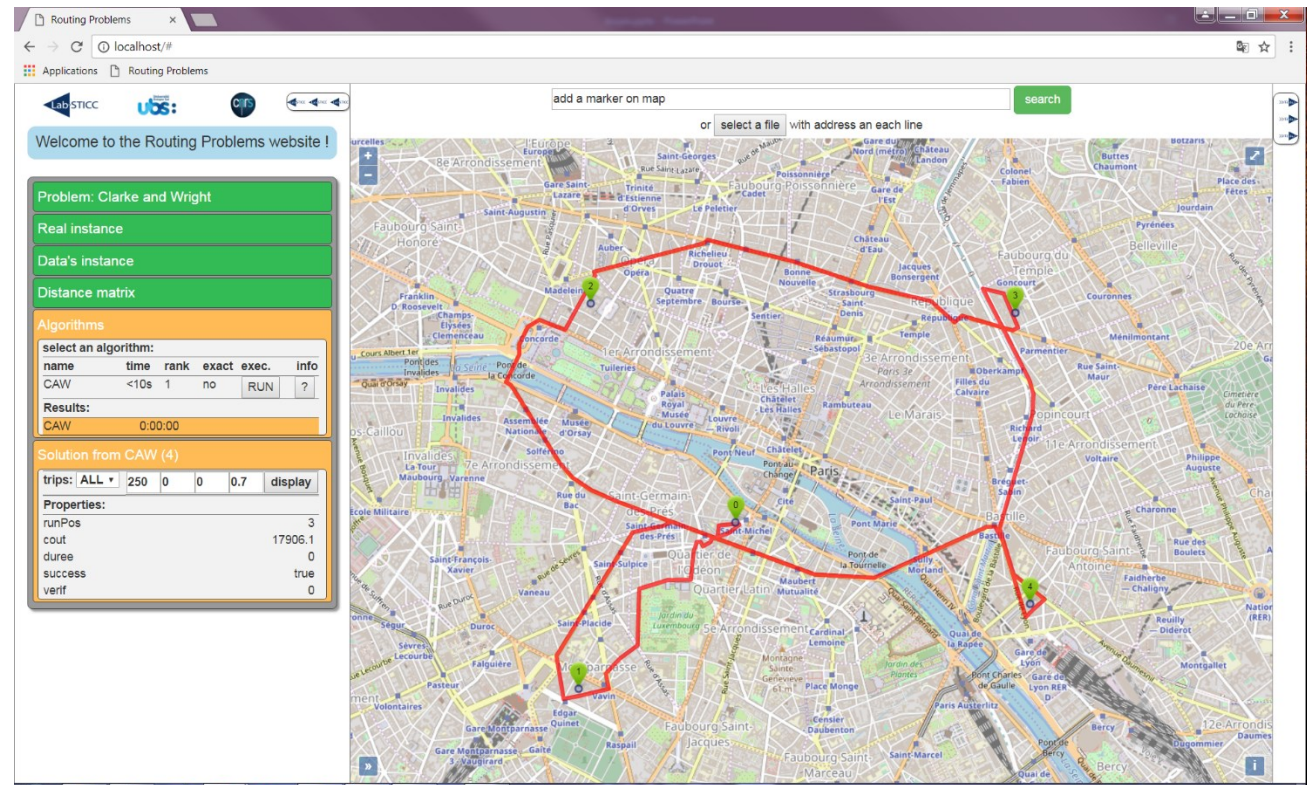
name	time	rank	exact	exec.	info
CAW	<10s	1	no	RUN	?

trips	ALL	250	0	0	0.7	display
runPos						2
cout						21421.5
duree						0
success						true
verif						0

The main area of the website shows a map of Paris with three red paths highlighted, representing the result of the first iteration of the Clarke and Wright algorithm. The paths are labeled with numbers 1, 2, and 3. The map includes various landmarks and street names, such as 'Gare Saint-Lazare', 'Opéra', 'Louvre', and 'Eiffel Tower'.

Clarke and Wright, C++

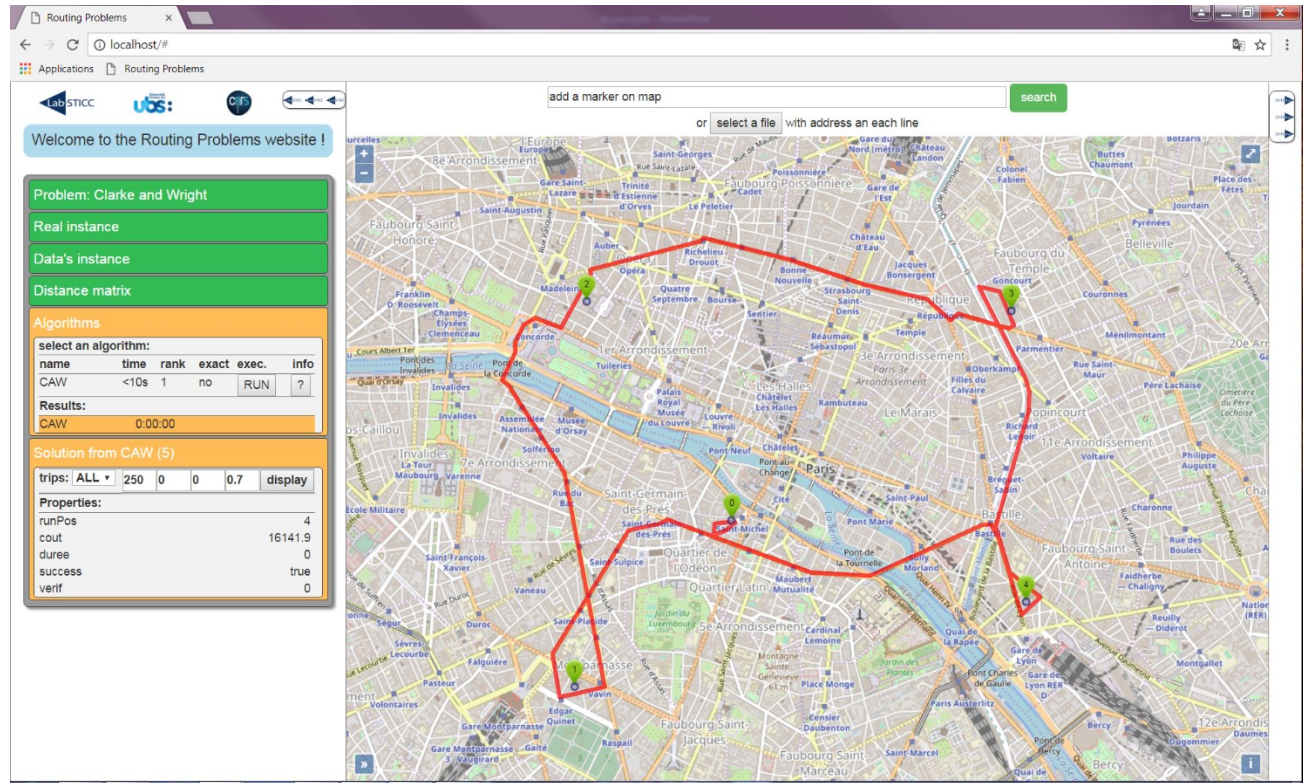
Here we can see the result
(2 paths) after the 2nd iteration.



Clarke and Wright, C++

Here we can see the result (1 path) after the 3rd iteration.

And this is the end of the algorithm because there is no more pair of paths to merge.

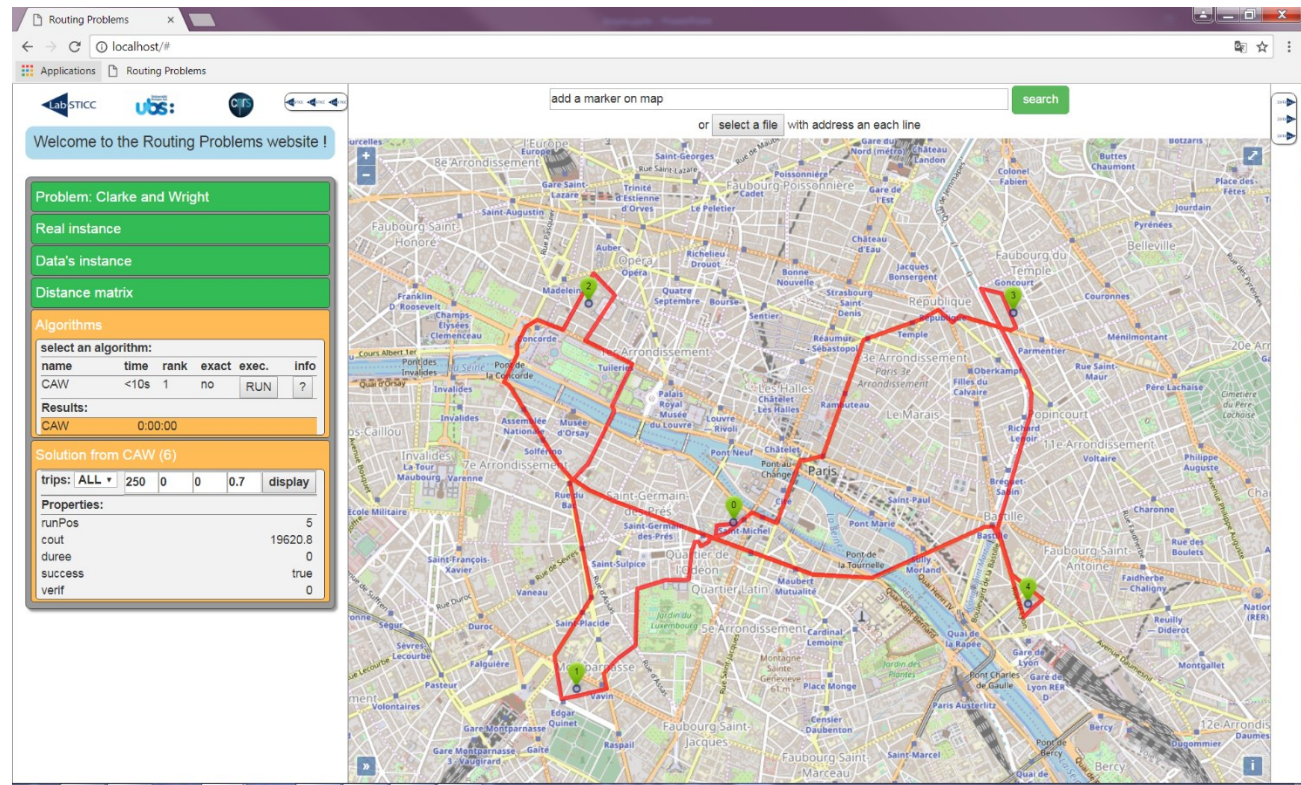


Clarke and Wright, C++

Now, imagine the delivery van has a capacity of 2 pizzas.

Because there are 4 customers, hence 4 pizzas, there must be at least 2 paths.

Here they are, with the vehicule capacity set to 4.



Clarke and Wright, C++

Now you will learn how to use the web-site to obtain this by yourself.

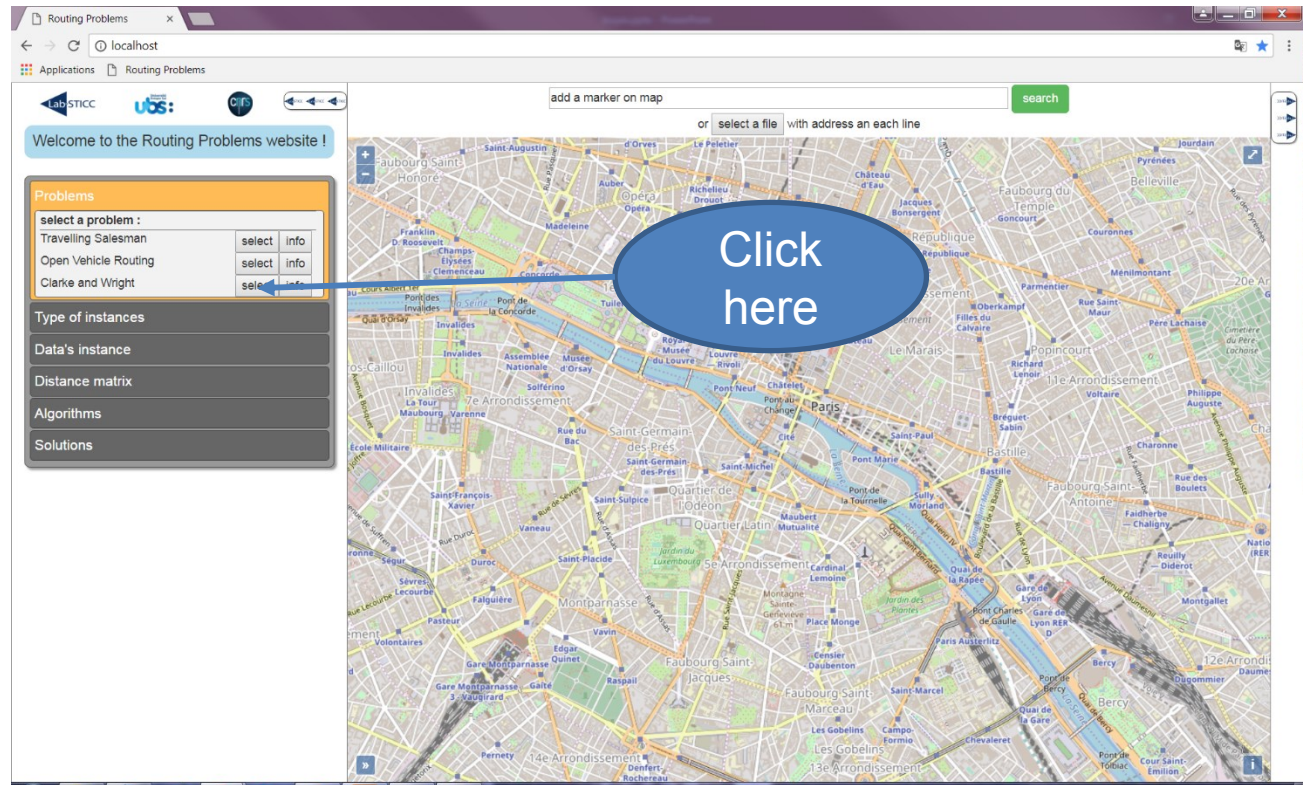
First open your preferred web-browser. I used Firefox.

Enter "localhost" as the URL where to load the web-services from.

Localhost means that your PC will provide the index.html file.

And this is possible because your PC runs the Apache HTTP server.

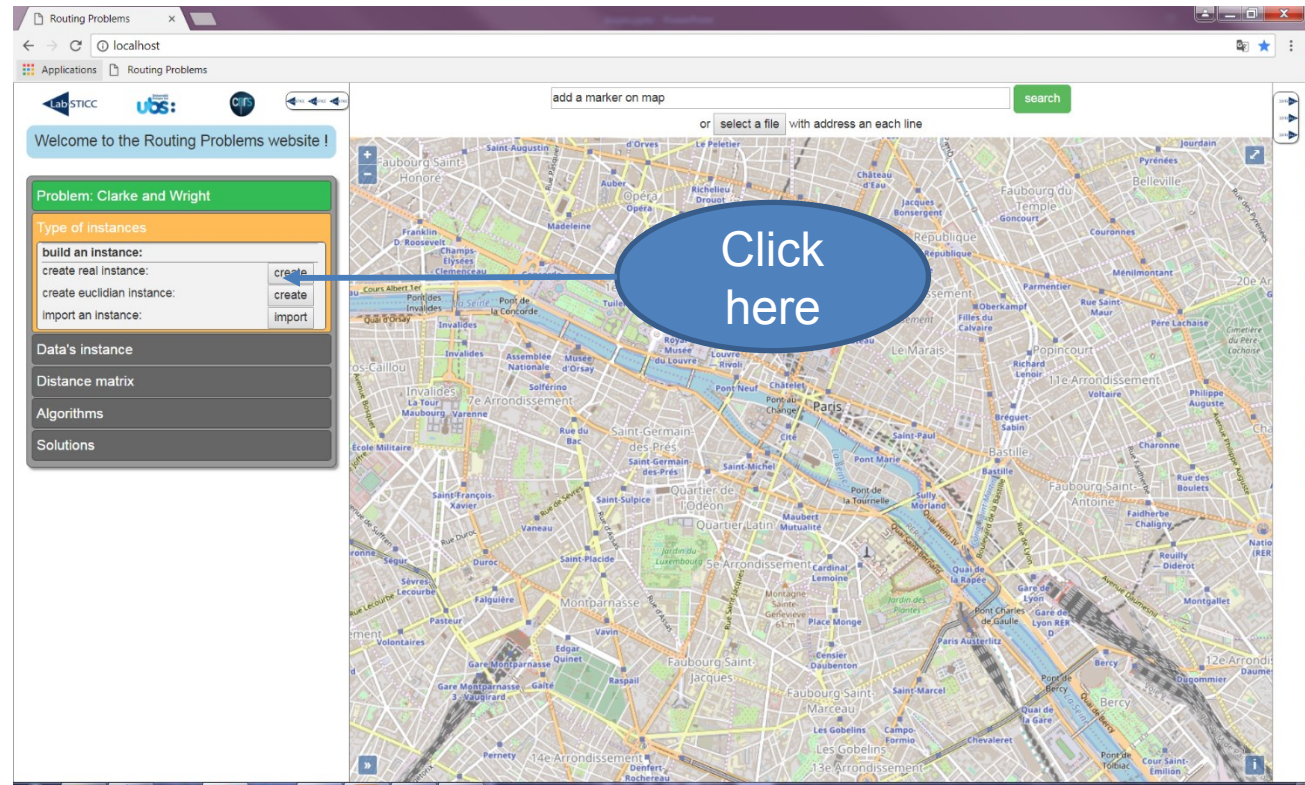
Select the CAW problem.



Clarke and Wright, C++

Select the way
"how to create the instance"

We'll use real GPS points.



Clarke and Wright, C++

Add randomly some points.

I added 10 points.

A list may appear on the right side on the window. It provides some tools to select and delete the points, as you will wish.

The screenshot shows the 'Routing Problems' website interface. On the left, there's a sidebar with a 'Welcome to the Routing Problems website !' message and a 'Problem: Clarke and Wright' section. This section includes a 'Real instance' tab and a 'Data's instance' section with fields for 'value of the courrant instance: Merges(integer)' (set to -1), 'Capacity(integer)' (set to 100), and a 'Depot(point)' section with 'Lon' and 'Lat' fields. Below these are buttons for 'load point', 'display', 'load GPS points from a file', 'current selection', 'load points', 'display: off', 'display in map', 'cancel', 'Save', 'Check', and 'OK'. At the bottom of the sidebar are buttons for 'Distance matrix', 'Algorithms', and 'Solutions'. The main area features a map of Paris with 10 points marked. A blue oval with the text 'To hide it click here' points to a button in the top right corner of the map area. Another blue oval with the text 'The list of points' points to a table on the right side of the interface. The table has columns for 'Id', 'Lon.', 'Lat.', and 'Del'. It lists 10 points with their coordinates and a 'Del' checkbox.

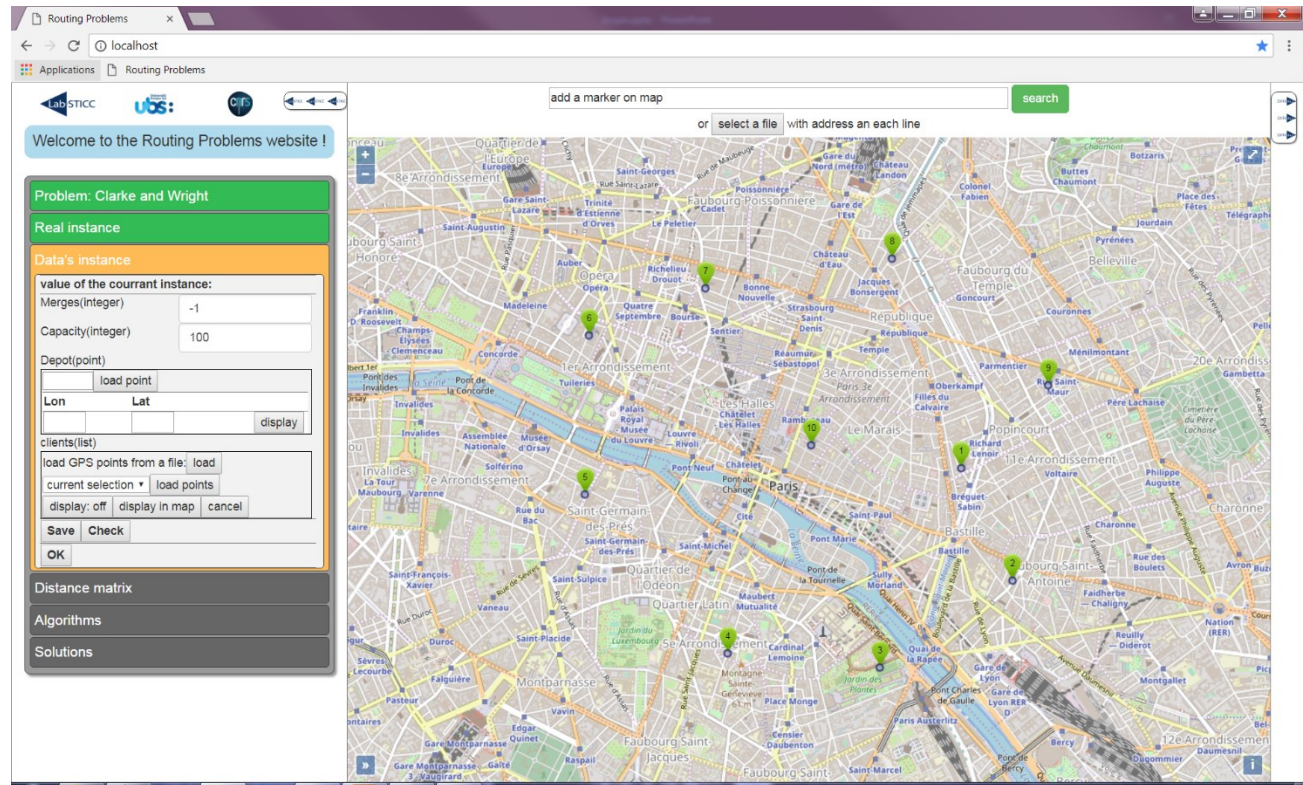
Id	Lon.	Lat.	Del
✓ 1	2.3696	48.8578	X
✓ 2	2.3749	48.8502	X
✓ 3	2.3613	48.8444	X
✓ 4	2.3458	48.8454	X
✓ 5	2.3313	48.8560	X
✓ 6	2.3317	48.8667	X
✓ 7	2.3436	48.8698	X
✓ 8	2.3626	48.8718	X
✓ 9	2.3785	48.8633	X
✓ 10	2.3543	48.8593	X

Clarke and Wright, C++

Hide the list of points.

You may drag the map to center the display on the point which will be the depot.

My depot will be point # 10



Clarke and Wright, C++

Merges is the number of iterations we allow CAW to execute:

- 1 means infinity (optimum)
- 0 means (keep the initial sol)
- n means n iterations

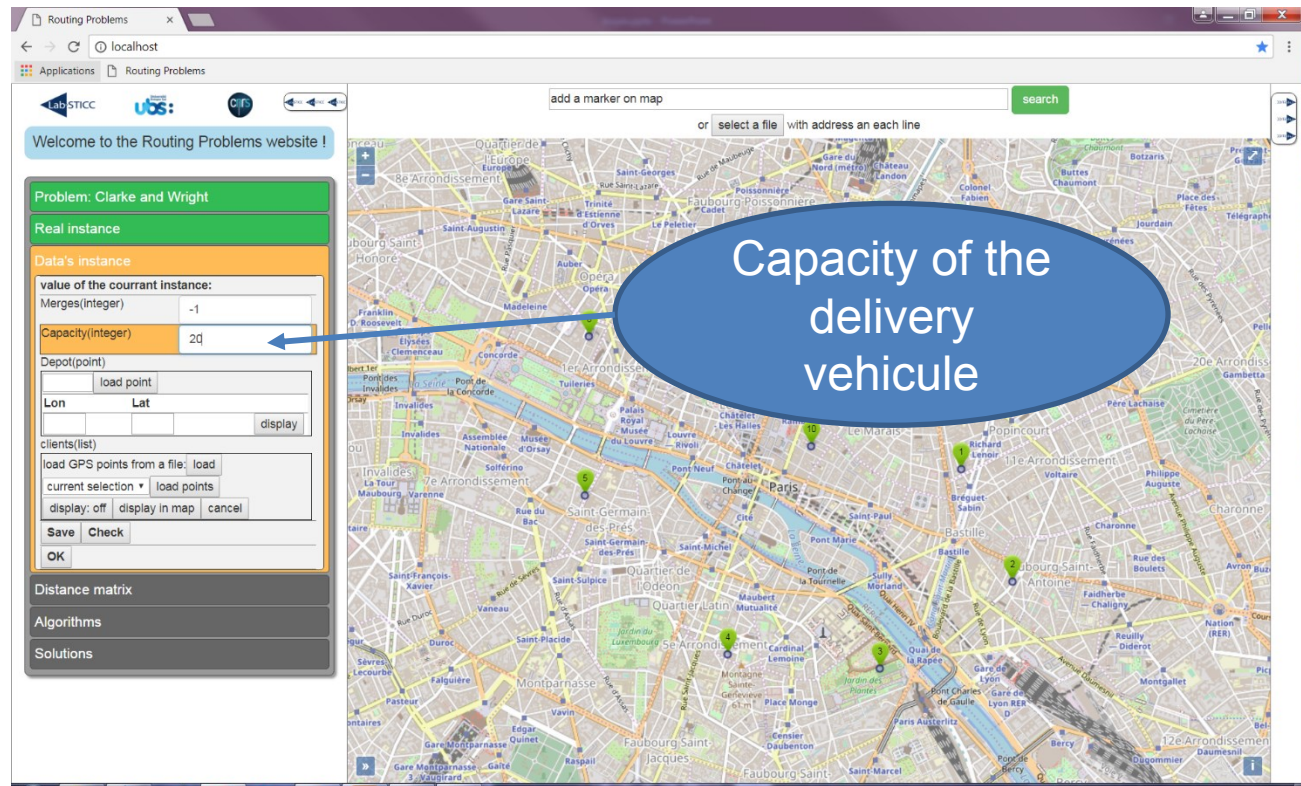
For now, keep the default value (-1)

The screenshot shows a web browser window with the URL 'localhost'. The page title is 'Routing Problems'. The main content area is titled 'Welcome to the Routing Problems website !'. On the left, there is a sidebar with a green header 'Problem: Clarke and Wright'. Below this, there is a 'Real instance' section with a 'Data's instance' sub-section. In the 'Data's instance' section, the 'value of the courrant instance:' is set to '-1' (highlighted by a blue arrow and a blue oval with the text 'Number of iterations'). The 'Capacity(Integer)' is set to '100'. There is a 'Depot(point)' section with a 'load point' button and a 'display' button. Below that is a 'clients(list)' section with a 'load GPS points from a file:' button, a 'current selection' dropdown, and a 'load points' button. There are also 'display: off', 'display in map', and 'cancel' buttons. At the bottom of the sidebar are 'Save', 'Check', 'OK', 'Distance matrix', 'Algorithms', and 'Solutions' buttons. The main content area on the right is a map of Paris with a search bar at the top. The search bar has a placeholder 'add a marker on map' and a 'search' button. Below the search bar, there is a link 'select a file with address an each line'. The map shows various districts of Paris, including the 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, 8th, 9th, 10th, 11th, 12th, 13th, 14th, 15th, 16th, 17th, 18th, 19th, 20th, and 21st. A blue arrow points from the 'Number of iterations' text to the '-1' value in the 'Data's instance' section.

Clarke and Wright, C++

Set the capacity.

I used 20 to indicate that the pizza shop's vehicle can deliver a maximum of 20 pizzas without returning back to the shop.



Clarke and Wright, C++

Select which point will be the depot.

I have chosen the point # 10.

The screenshot shows the 'Routing Problems' website interface. The left sidebar contains a form for problem configuration. The main area displays a map of Paris with several points marked. Annotations in blue ovals provide instructions:

- 1/ Give the identity of the depot**: Points to the 'Depot(point)' field in the form, which has '10' entered.
- 2/ Click on « load point »**: Points to the 'load point' button in the form.
- 3/ GPS coords will appear there**: Points to the 'Lon' and 'Lat' input fields in the form, which contain the coordinates '2.3543' and '48.859' respectively.

The form fields include:

- Problem: Clarke and Wright
- Real instance
- Data's instance
 - value of the courant instance: -1
 - Merges(integer): 20
 - Capacity(integer): 20
 - Depot(point): 10
 - Lon: 2.3543
 - Lat: 48.859
 - clients(list):
 - load GPS points from a file: load
 - current selection: load points
 - display: off
 - display in map: cancel
 - Save
 - Check
 - OK
- Distance matrix
- Algorithms
- Solutions

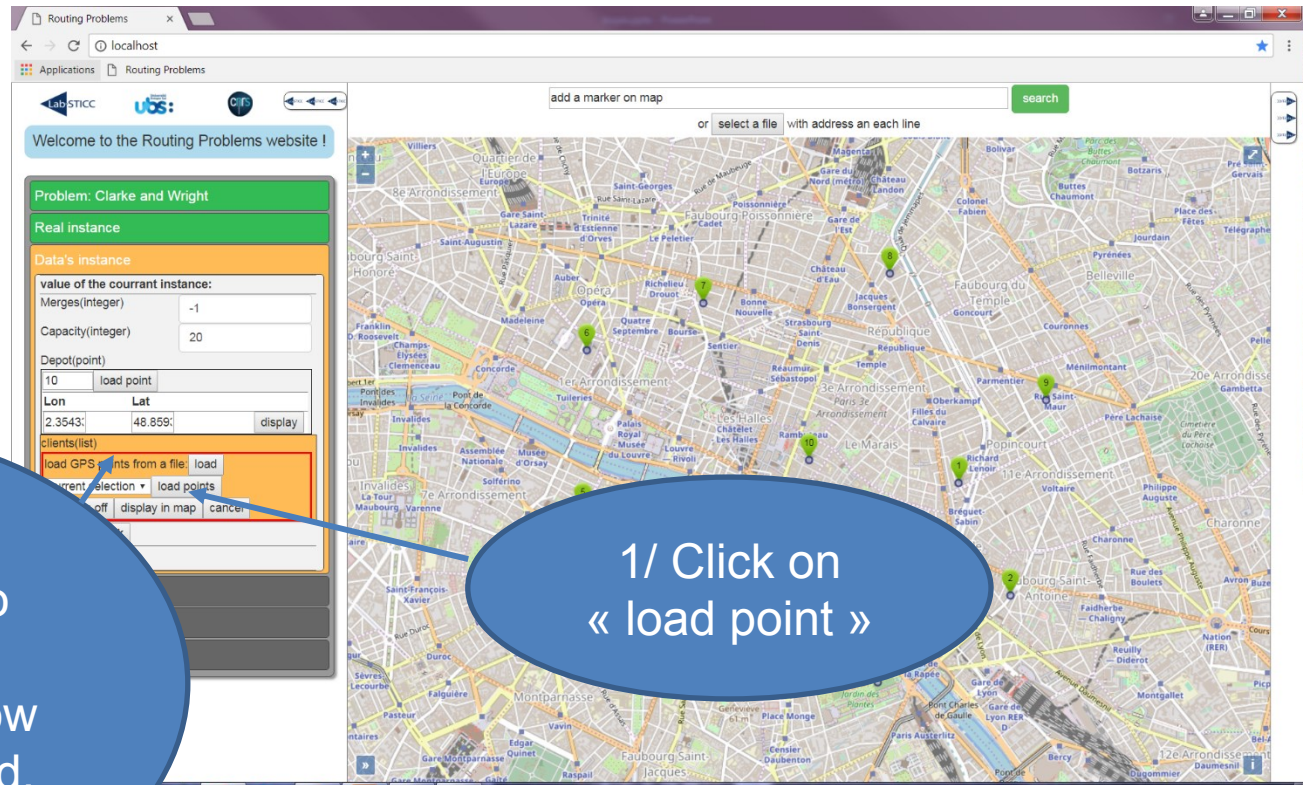
Clarke and Wright, C++

Select the list of points to use for CAW.

I decided to use the points from the list of points who was displayed on the right. Its points are all the ones displayed on the map. They are numbered from 1 to 10.

2/ Unfortunately it doesn't seem to work ...

And the sub-window is highlighted in red. What does it mean ?



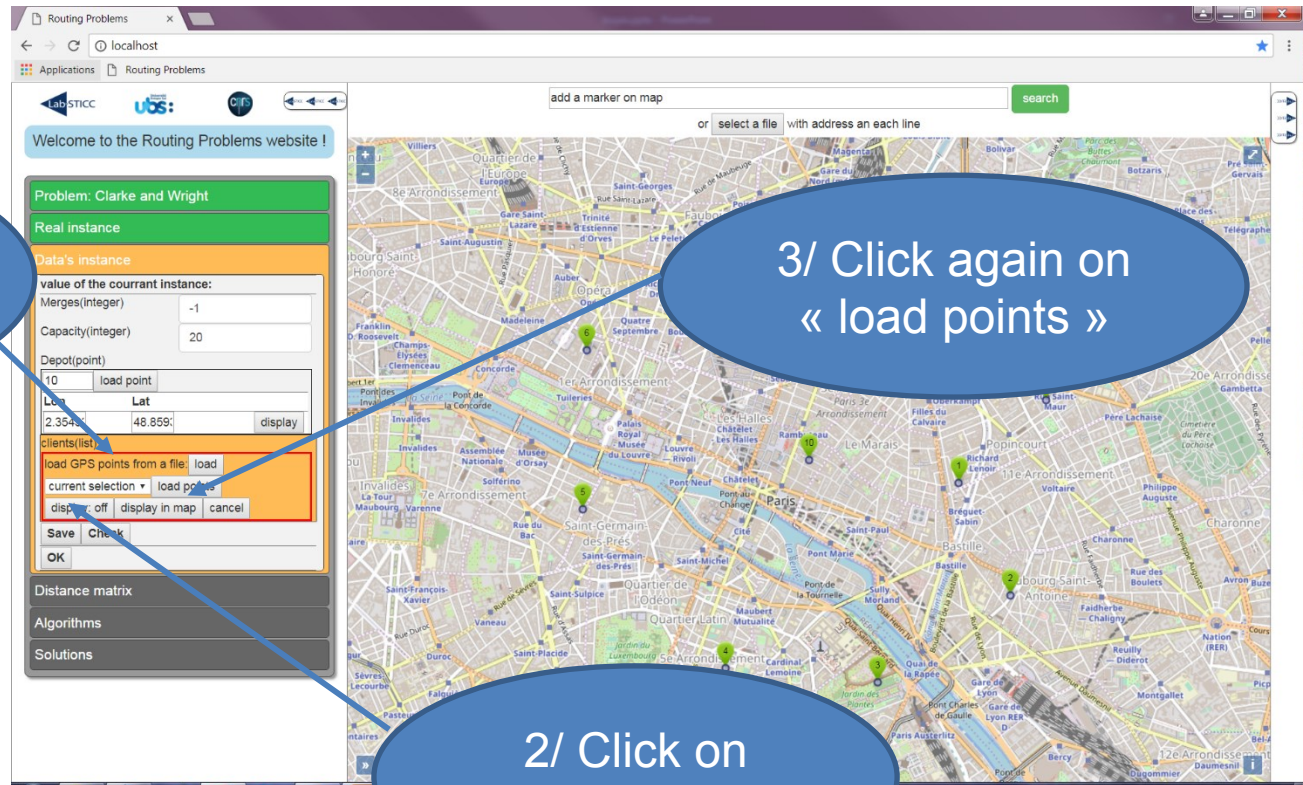
Clarke and Wright, C++

Select the list of points to use for CAW.

1/ Don't be afraid.
There is no error.

3/ Click again on
« load points »

2/ Click on
« display off »



Clarke and Wright, C++

Select the list of points to use for CAW.

Now you can see them.

The screenshot shows the 'Routing Problems' website interface. The left sidebar contains the following sections:

- Problem: Clarke and Wright**
- Real instance**
- Data's instance**
 - value of the courant instance: -1
 - Capacity(integer): 20
 - Depot(point): 10
 - load point:
 - clients(list):

Lon	Lat	Goods
2.3696	48.857	
2.3748	48.850	
2.3613	48.844	
2.3468	48.845	
2.3312	48.856	
2.3316	48.866	
2.3435	48.869	
2.3625	48.871	
2.3785	48.863	
2.3543	48.859	

The main area displays a map of Paris with several points marked. A search bar at the top right contains the text 'add a marker on map' and 'select a file with address an each line'. The map shows various districts and landmarks, with points numbered 1 through 10.

Clarke and Wright, C++

Select the list of points to use for CAW.

When all is green, all is good !

The screenshot shows a web application titled "Routing Problems" running on localhost. The interface includes a search bar at the top right with the text "add a marker on map" and a "search" button. Below the search bar, there is a map of Paris with several green markers placed on it. On the left side, there is a form for configuring the problem. The form has a section titled "Problem: Clarke and Wright" and a "Real instance" section. Under "Data's instance", there are input fields for "value of the courant instance:" (set to -1) and "Capacity(integer)" (set to 20). Below these, there is a "Depot(point)" section with a table for "clients(list)". The table has columns for "Lon", "Lat", and "Goods". The "Goods" column contains green checkboxes, and a blue arrow points to one of them. At the bottom of the form, there are "Save" and "Check" buttons.

Type in the quantities (of pizzas), and click on « check ».

Lon	Lat	Goods
2.3696	48.857	☐
2.3748	48.850	☐
2.3613	48.844	☐
2.3458	48.845	☐
2.3312	48.856	☐
2.3316	48.866	☐
2.3435	48.869	☐
2.3625	48.871	☐
2.3785	48.863	☐
2.3543	48.859	☐

Clarke and Wright, C++

Select the list of points to use for CAW.

The screenshot shows a web application titled "Routing Problems" running on localhost. The interface is divided into two main sections. On the left is a "Real instance" panel, and on the right is a map of Paris.

Real instance panel:

- Data's instance:**
 - value of the courant instance: (empty)
 - Merges(integer): -1
 - Capacity(integer): 20
 - Depot(point): 10
 - load point: (empty)
 - Lon: 2.3543, Lat: 48.8591, display: (empty)
- clients(list):**
 - load GPS points from a file: load
 - current selection: (empty)
 - load points: (empty)
 - display: on, display in map, cancel
 - Table with 4 columns: Lon, Lat, Goods, and an empty column for selection.

Lon	Lat	Goods	
2.3696	48.8571	1	
2.3748	48.8501	1	
2.3613	48.8441	1	
2.3458	48.8451	1	
2.3312	48.8561	1	
2.3316	48.8661	1	
2.3435	48.8691	1	
2.3625	48.8711	1	
2.3785	48.8631	1	
2.3543	48.8591	0	
 - Buttons: Save, Check, OK
- Distance matrix**
- Algorithms**
- Solutions**

Map section:

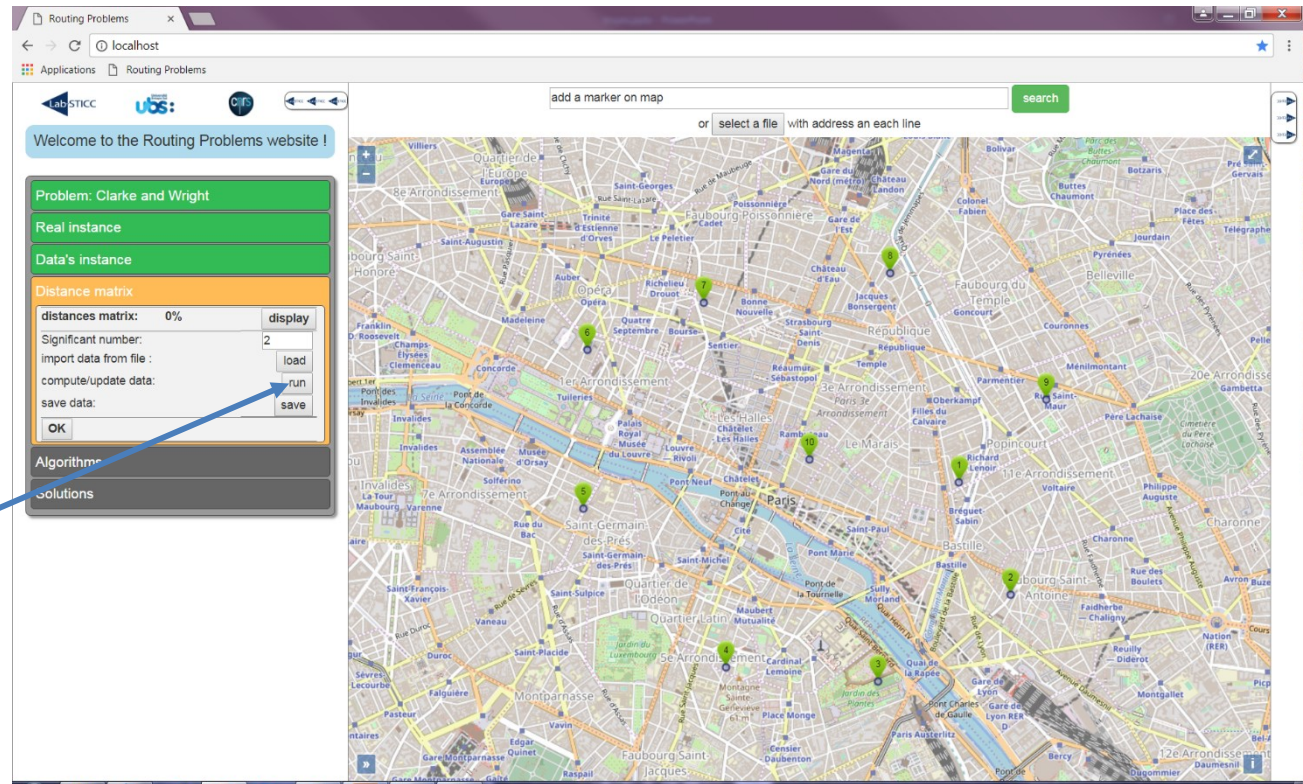
- Search bar: "add a marker on map" or "select a file with address on each line"
- Map of Paris with several points marked with numbers 1 through 10.

To see the « OK » button you may need to use the slider.

Click on
« OK »

Clarke and Wright, C++

Now it is time to compute the distances matrix.



Click on
« run »

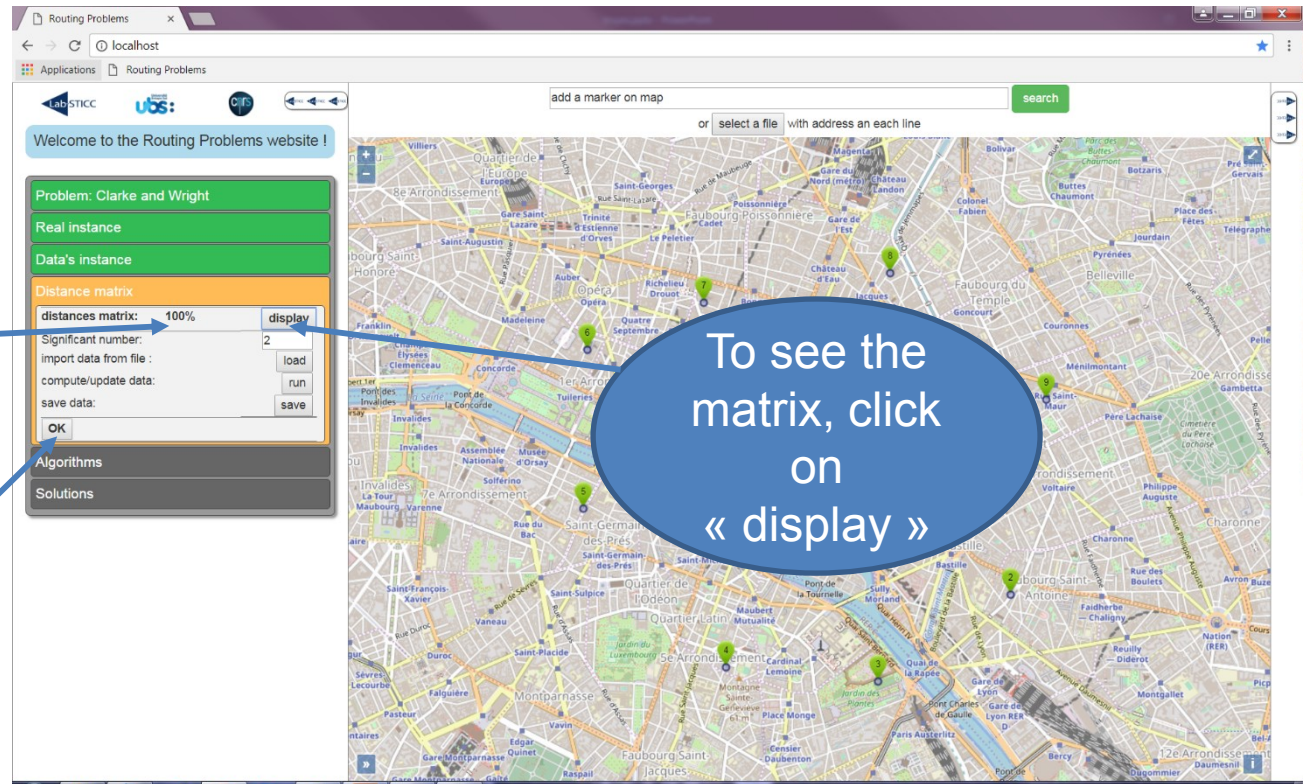
Clarke and Wright, C++

Now it is time to compute the distances matrix.

When computed 100% is displayed

To proceed click on « OK »

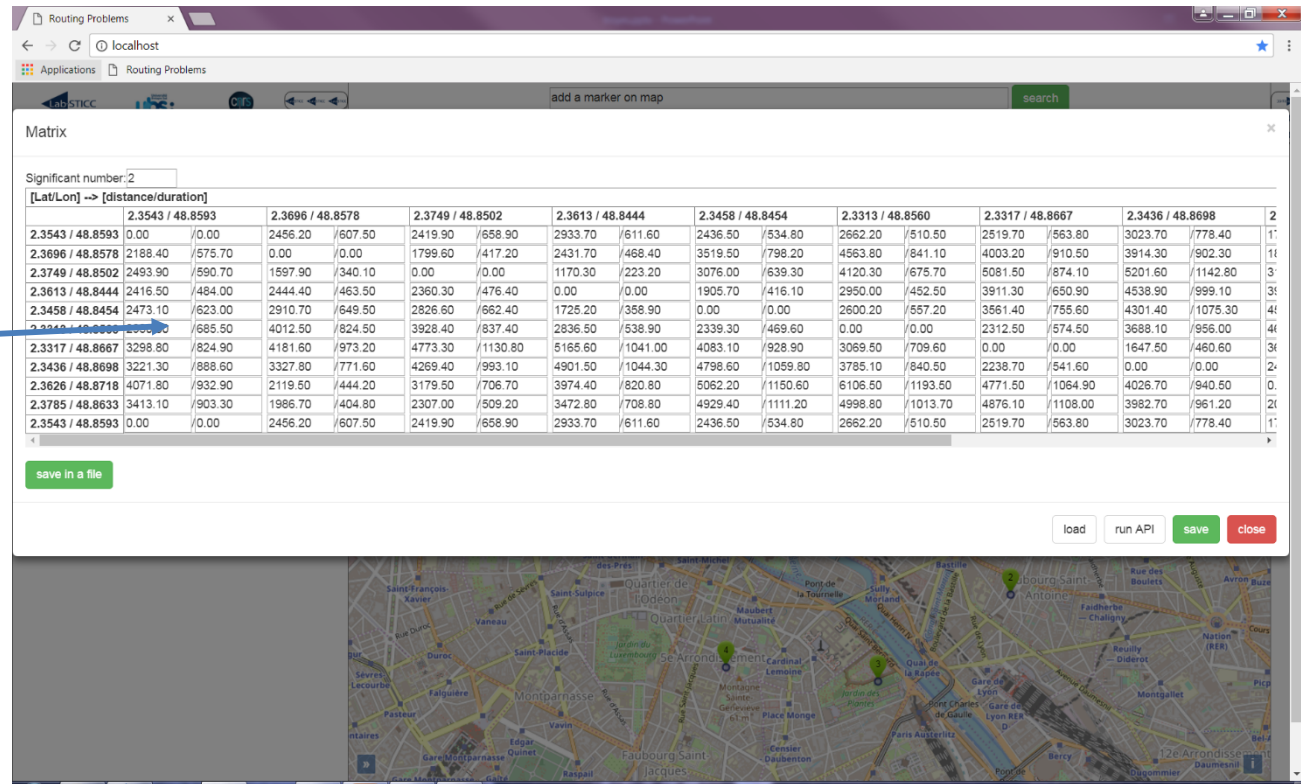
To see the matrix, click on « display »



Clarke and Wright, C++

Now it is time to compute the distances matrix.

Here it is.



The screenshot shows a web application titled "Routing Problems" running on a local host. It features a "Matrix" section with a table of distances and durations between 10 locations in Paris. A blue callout bubble points to the table with the text "Here it is." Below the table is a "save in a file" button. At the bottom of the application is a map of Paris with markers at the locations defined in the matrix. Control buttons for "load", "run API", "save", and "close" are located at the bottom right of the interface.

Significant number: 2

[Lat/Lon] --> [distance/duration]

	2.3543 / 48.8593	2.3696 / 48.8578	2.3749 / 48.8502	2.3613 / 48.8444	2.3458 / 48.8454	2.3313 / 48.8560	2.3317 / 48.8667	2.3436 / 48.8698	2
2.3543 / 48.8593	0.00 / 0.00	2456.20 / 607.50	2419.90 / 658.90	2933.70 / 611.60	2436.50 / 534.80	2662.20 / 510.50	2519.70 / 563.80	3023.70 / 778.40	1'
2.3696 / 48.8578	2188.40 / 575.70	0.00 / 0.00	1799.60 / 417.20	2431.70 / 468.40	3519.50 / 798.20	4563.80 / 841.10	4003.20 / 910.50	3914.30 / 902.30	11'
2.3749 / 48.8502	2493.90 / 590.70	1597.90 / 340.10	0.00 / 0.00	1170.30 / 223.20	3076.00 / 639.30	4120.30 / 675.70	5081.50 / 874.10	5201.60 / 1142.80	3'
2.3613 / 48.8444	2416.50 / 484.00	2444.40 / 463.50	2360.30 / 476.40	0.00 / 0.00	1905.70 / 416.10	2950.00 / 452.50	3911.30 / 650.90	4538.90 / 999.10	3'
2.3458 / 48.8454	2473.10 / 623.00	2910.70 / 649.50	2826.60 / 662.40	1725.20 / 358.90	0.00 / 0.00	2600.20 / 557.20	3561.40 / 755.60	4301.40 / 1075.30	4'
2.3313 / 48.8560	2493.90 / 590.70	685.50 / 4012.50	824.50 / 3928.40	837.40 / 2836.50	538.90 / 2339.30	469.60 / 0.00	0.00 / 2312.50	574.50 / 3688.10	4'
2.3317 / 48.8667	3298.80 / 824.90	4181.60 / 973.20	4773.30 / 1130.80	5165.60 / 1041.00	4083.10 / 928.90	3069.50 / 709.60	0.00 / 0.00	1647.50 / 460.60	3'
2.3436 / 48.8698	3221.30 / 888.60	3327.80 / 771.60	4269.40 / 993.10	4901.50 / 1044.30	4798.60 / 1059.80	3785.10 / 840.50	2238.70 / 541.60	0.00 / 0.00	2'
2.3785 / 48.8633	3413.10 / 903.30	1966.70 / 404.80	2307.00 / 509.20	3472.80 / 708.80	4929.40 / 1111.20	4998.80 / 1013.70	4876.10 / 1108.00	3982.70 / 961.20	2'
2.3543 / 48.8593	0.00 / 0.00	2456.20 / 607.50	2419.90 / 658.90	2933.70 / 611.60	2436.50 / 534.80	2662.20 / 510.50	2519.70 / 563.80	3023.70 / 778.40	1'

save in a file

load run API save close

Finally you will choose an algorithm and run it with the instance we have created.



Click on
« run »



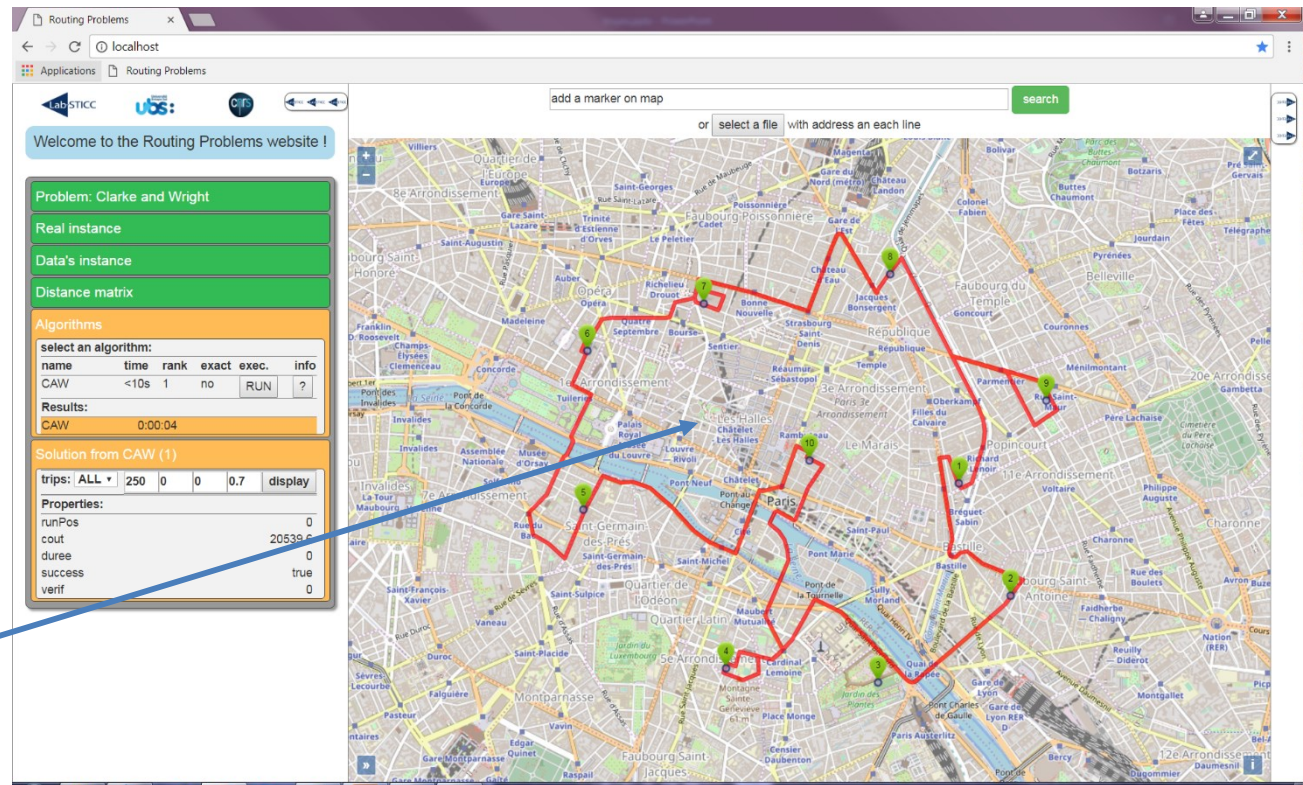
24

Clarke and Wright, C++

The result will depend on the instance and on the chosen algorithm.

Here is mine.

When finished, the paths are displayed on the map.



Clarke and Wright, C++

Before adding your own web-service, take a few minutes to check your ability to play with CAW.

I suggest you to try it with different values.

The screenshot shows the 'Routing Problems' website interface. The main panel is titled 'Problem: Clarke and Wright' and contains a 'Real instance' section. The 'Data's instance' section includes input fields for 'value of the courrant instance:' (set to -1), 'Capacity(integer)' (set to 20), and 'Depot(point)' (set to 10). Below these is a 'clients(list)' section with a table of client data. The table has columns for 'Lon', 'Lat', and 'Goods'. The 'Goods' column is currently empty. A blue oval callout points to the 'Goods' column with the text 'Choose another depot'. Another blue oval callout points to the 'Capacity' field with the text 'Capacities 100, 10, 5, 3, 1'. A third blue oval callout points to the 'Merges' field with the text 'Merges 0, 1, 2, 3'. A fourth blue oval callout points to the 'Buy more pizzas' button with the text 'Buy more pizzas 😊'. The background of the interface is a map of Paris.

Lon	Lat	Goods
2.3696	48.857	
2.3748	48.850	
2.3613	48.844	
2.3458	48.845	
2.3312	48.856	
2.3316	48.866	
2.3435	48.869	
2.3625	48.871	
2.3785	48.863	
2.3543	48.859	

Buy more pizzas 😊

Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

C++

The Clarke and Wright algorithm.

Modifying the CAW and testing it.

Adding your own strategy to the CAW.

How to debug ?



Clarke and Wright, C++

Before coming to the Spring School we have been in touch and I gave you some files to install on your PC.

We'll have now a close look to some of them.

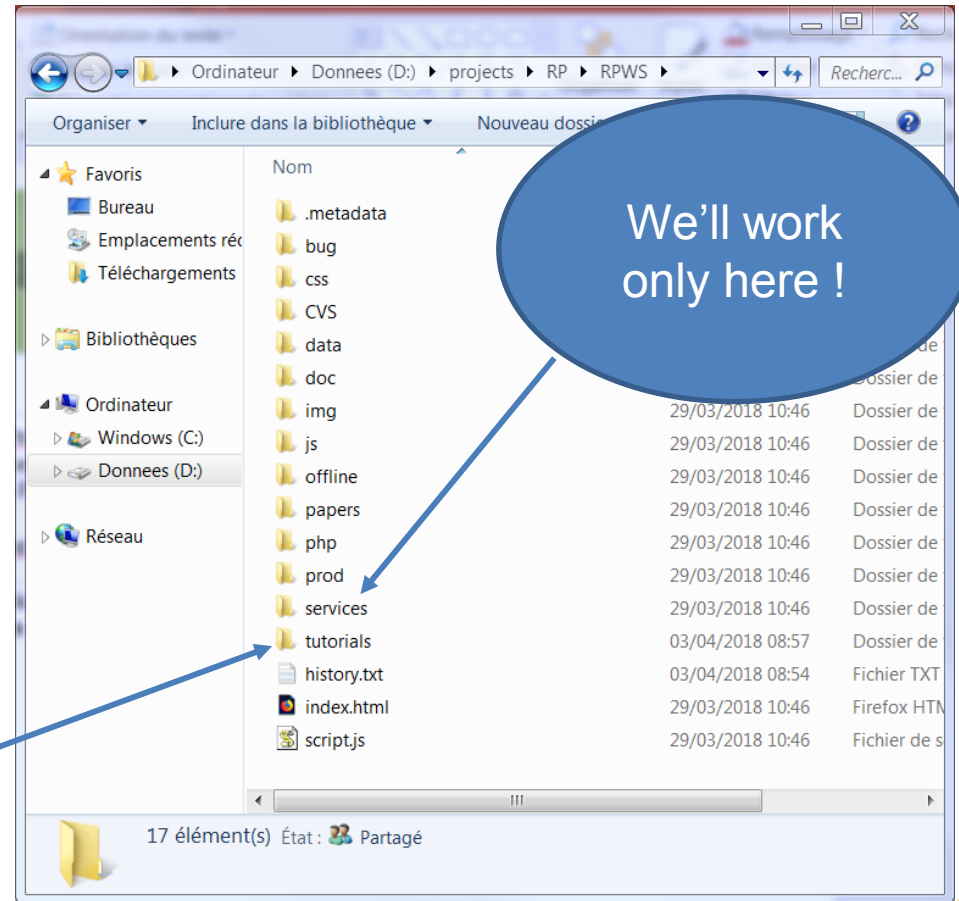
From now `<dir>` will be the location on your hard drives where you have installed the files.

My `<dir>` = `D:\projects\RP\RPWS`

`<dir>\services` contains all the web-services

PDF and PPT files (this one too) for all tutorials can be found at

`<dir>\tutorials`



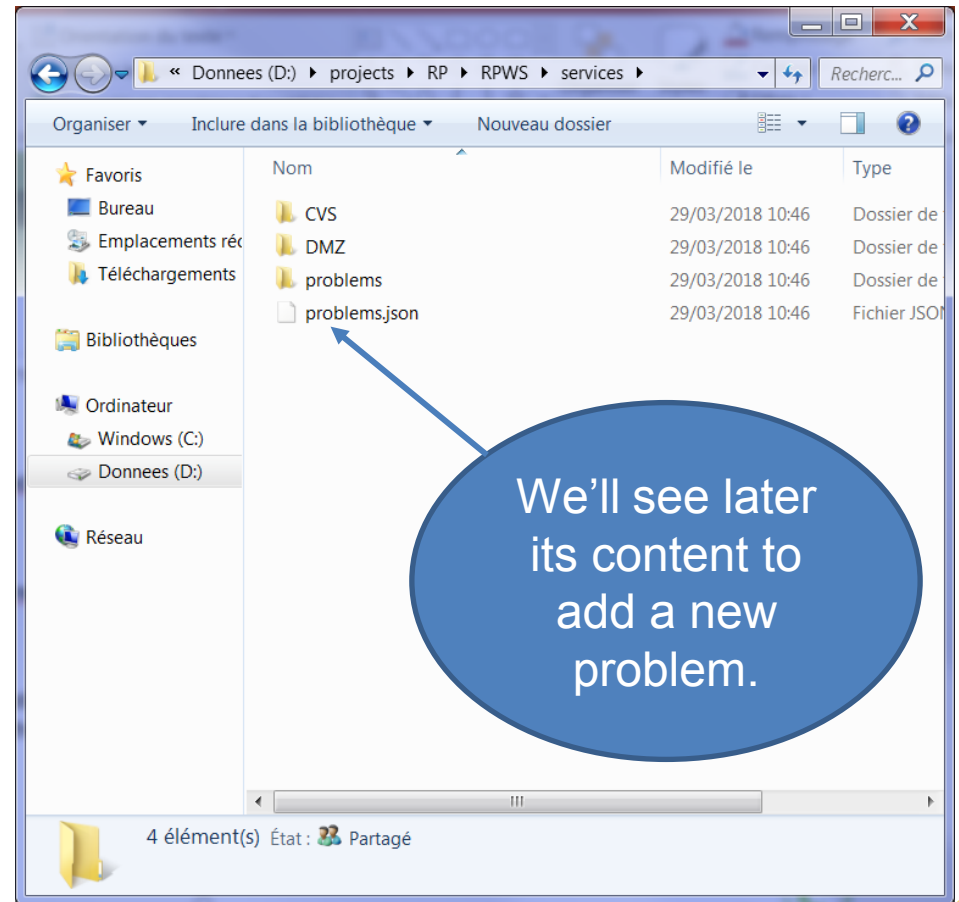
Clarke and Wright, C++

Let's have a look at `<dir>\services`

`DMZ` is the location of exchanged data between the client browser and the HTTP server.

`problems` is the directory where are stored all the algorithms for all the problems.

`problems.json` is the description of the list of problems handled by the web-site.



Clarke and Wright, C++

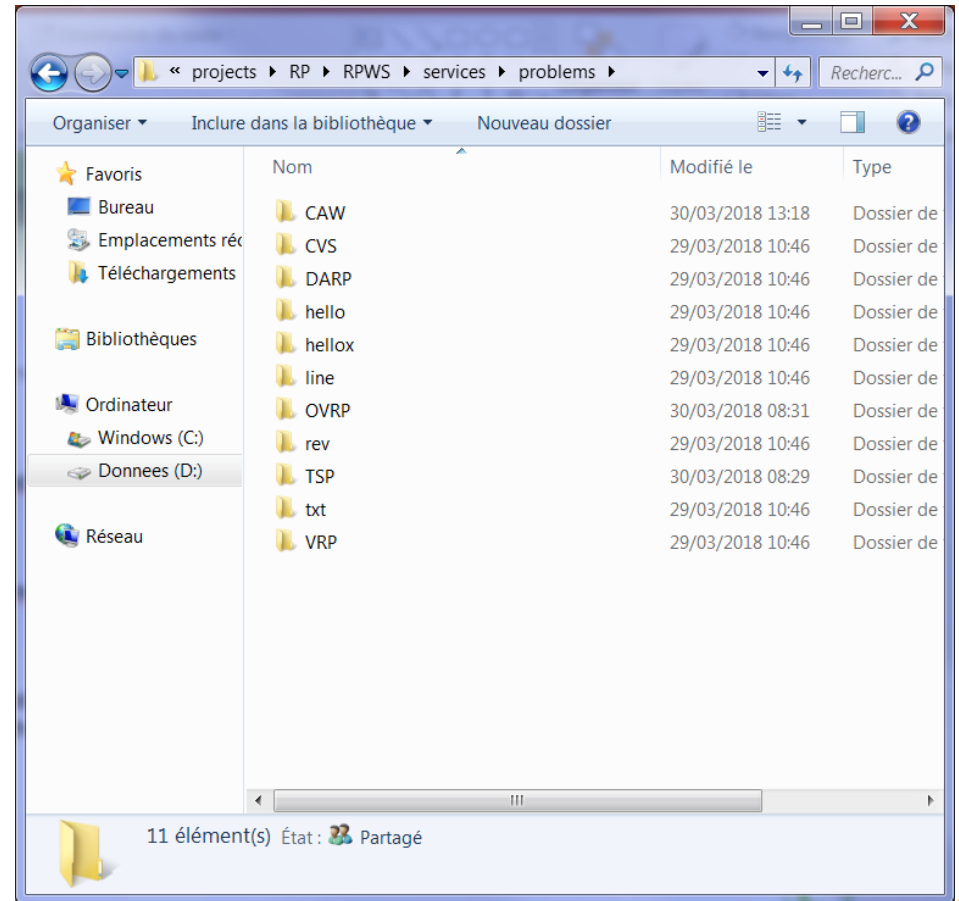
Let's have a look at <dir>\services\problems

Each directory is a web-service.

With the latest version of our web-site, only

CAW (Clarke and Wright)
OVRP (Open Vehicule Routing)
and
TSP (Traveling Salesman)
are operational.

Forget all the others !



Clarke and Wright, C++

Let's have a look at

`<dir>\services\problems\CAW`

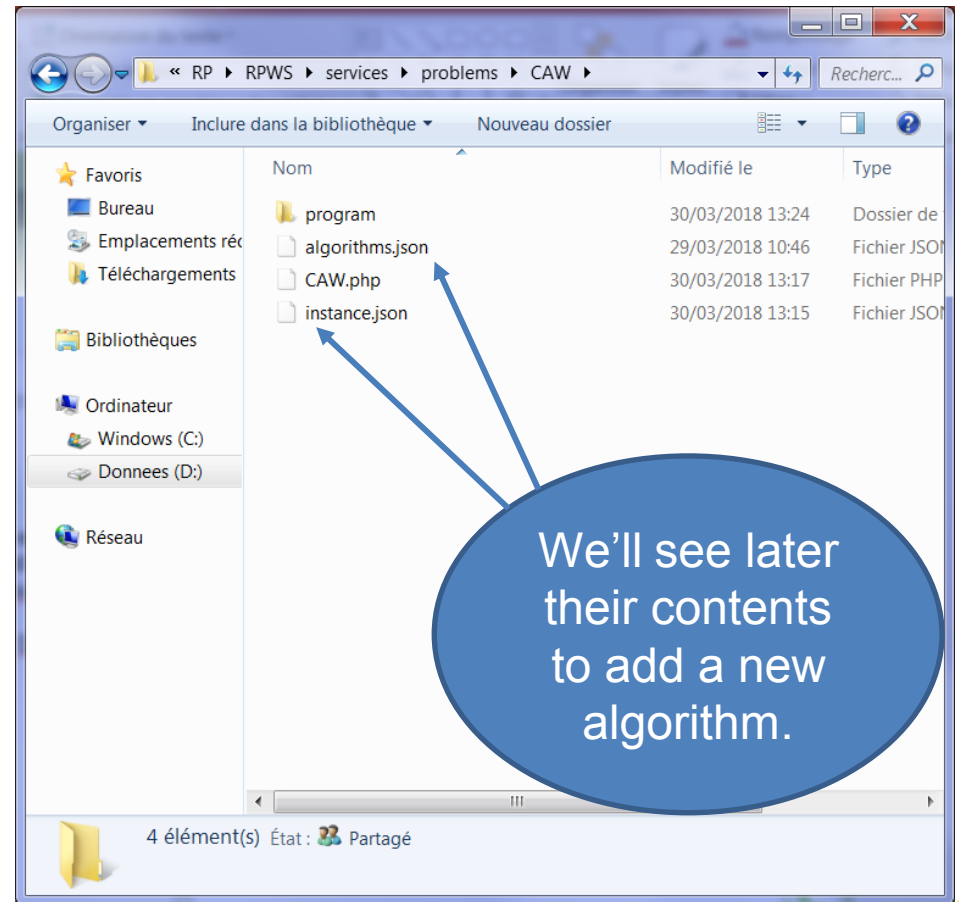
Each problem has the same structure !

`program` contains executable algorithms
implemented with the programming
languages of your choice

`algorithms.json` describes the set of
available algorithms for this problem
for the web-browser.

`CAW.php` is the interface between the
« network-world » and your executable.
This file is responsible for
1) converting the network data into
« executable-compatible » data,
2) calling the executable, and
3) convert back the results to
« network-format ».

`instance.json` describes what is an
instance for the web browser.



Clarke and Wright, C++

Let's have a look at

<dir>\services\problems\CAW\program

caw.cc C++ implementation of
all classes

caw.exe the executable

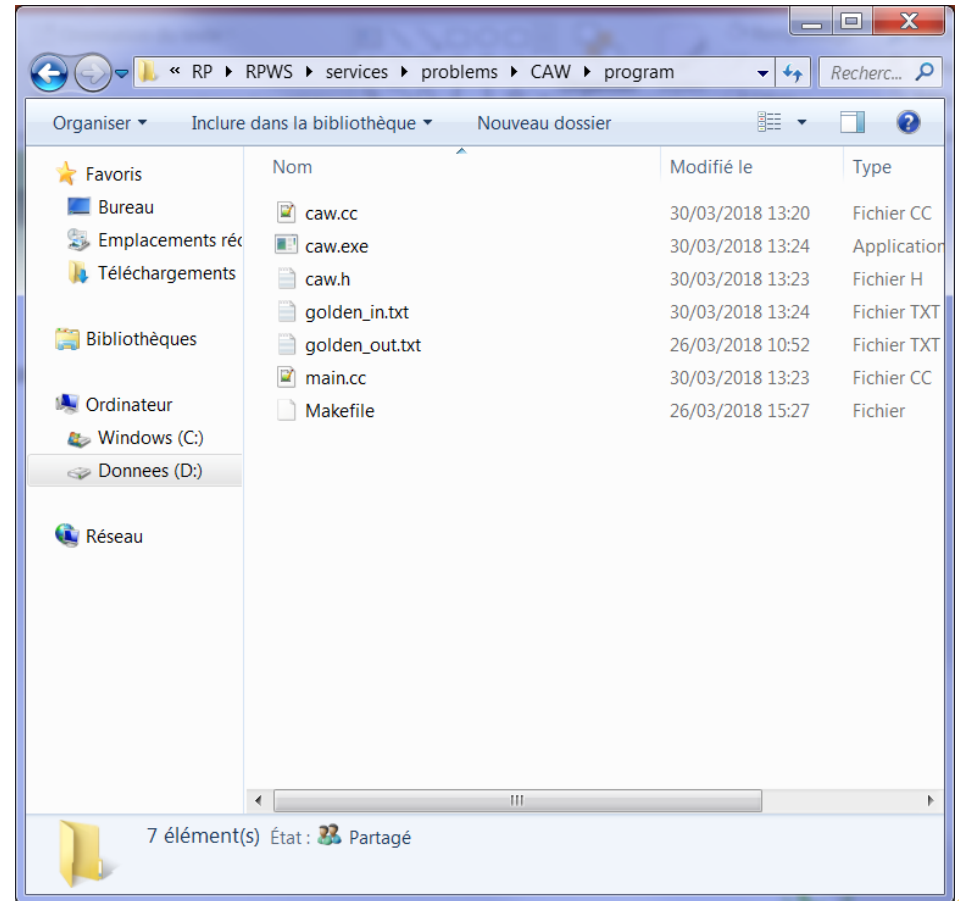
caw.h C++ classes header

golden_in.txt ref input file

golden_out.txt ref output file

main.cc the algorithm !!!!

Makefile automated make



Clarke and Wright, C++

Let's have a look at

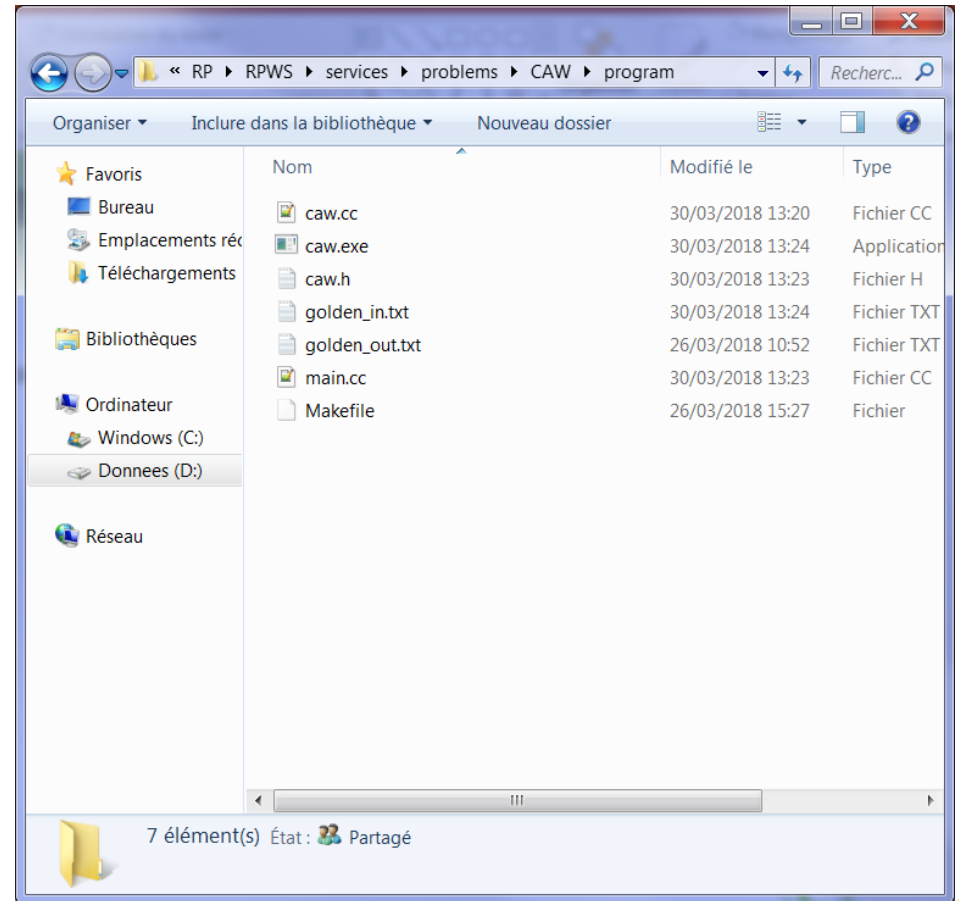
```
<dir>\services\problems\CAW\program
```

You should have installed a C++ development environment on your PC.

I suggested for Windows, CYGWIN or MINGW or to use Linux 😊

Whatever has been your choice

To clean	<code>make clean</code>
To build	<code>make</code>
To check	<code>make check</code>



Clarke and Wright, C++

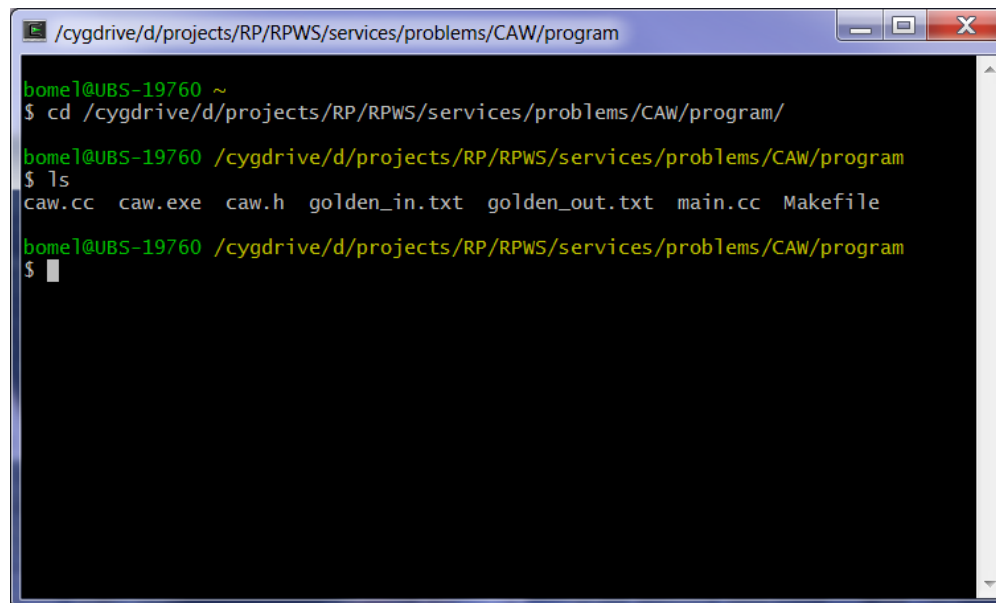
Let's have a look at

`<dir>\services\problems\CAW\program`

I suggest you take a few minutes to learn how to rebuild the CAW executable.

I will illustrate it with CYGWIN tools and `<dir> = /cygdrive/d/projects/RP/RPWS`

1/ go to the good location and list all the files



```
/cygdrive/d/projects/RP/RPWS/services/problems/CAW/program  
bome1@UBS-19760 ~  
$ cd /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program/  
bome1@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program  
$ ls  
caw.cc  caw.exe  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile  
bome1@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program  
$
```

Clarke and Wright, C++

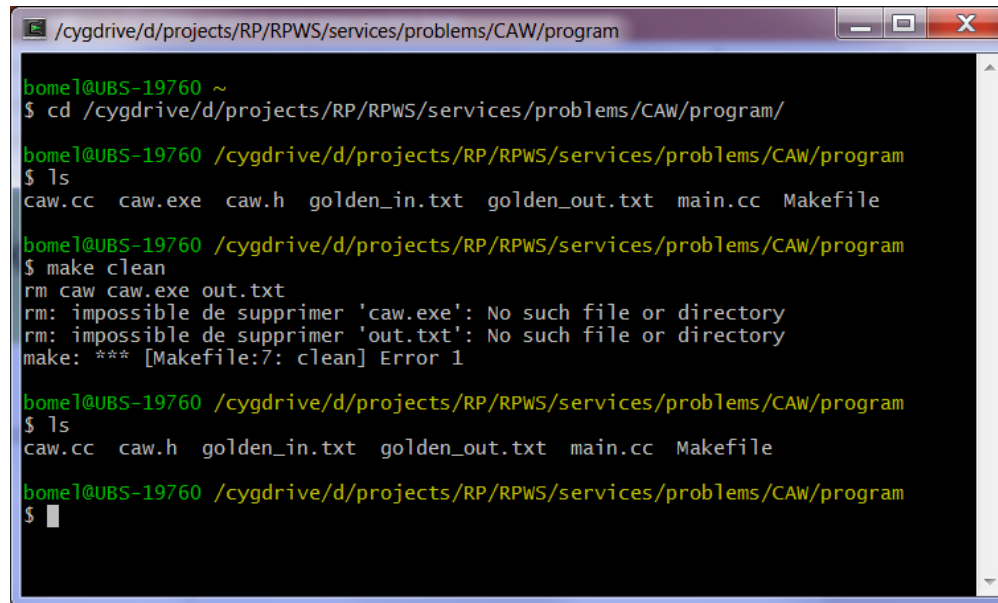
Let's have a look at

`<dir>\services\problems\CAW\program`

2/ restart from a fresh and clean state:

`make clean`

The `caw.exe` file should have been removed.



```
/cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
bome1@UBS-19760 ~
$ cd /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program/
bome1@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ ls
caw.cc  caw.exe  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile
bome1@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ make clean
rm caw caw.exe out.txt
rm: impossible de supprimer 'caw.exe': No such file or directory
rm: impossible de supprimer 'out.txt': No such file or directory
make: *** [Makefile:7: clean] Error 1
bome1@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ ls
caw.cc  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile
bome1@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$
```

Clarke and Wright, C++

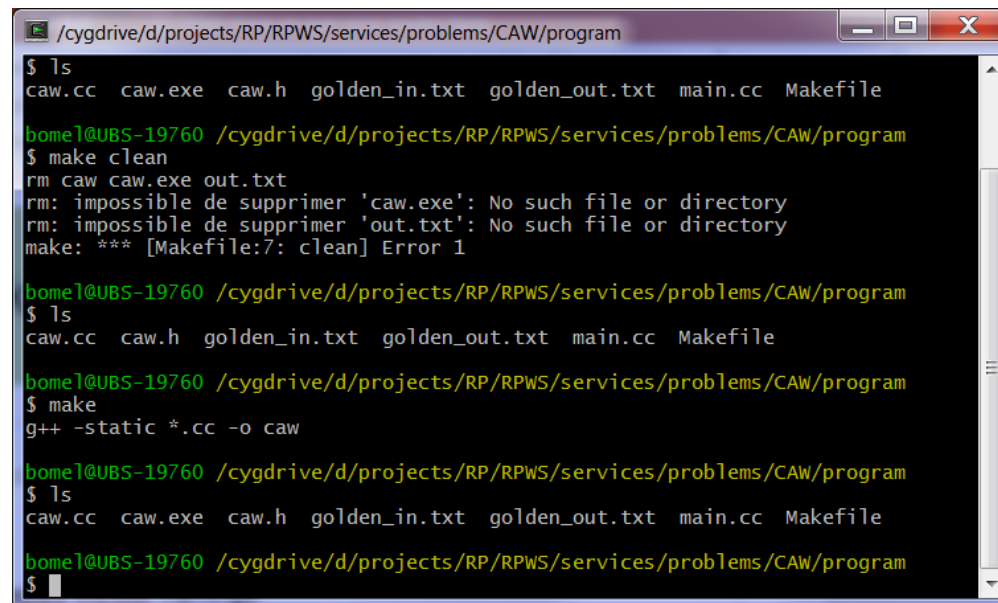
Let's have a look at

<dir>\services\problems\CAW\program

3/ rebuild:

make

The `caw.exe` file should have been created without warnings.



```
/cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ ls
caw.cc  caw.exe  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ make clean
rm caw caw.exe out.txt
rm: impossible de supprimer 'caw.exe': No such file or directory
rm: impossible de supprimer 'out.txt': No such file or directory
make: *** [Makefile:7: clean] Error 1

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ ls
caw.cc  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ make
g++ -static *.cc -o caw

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ ls
caw.cc  caw.exe  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$
```

Clarke and Wright, C++

Let's have a look at

<dir>\services\problems\CAW\program

4/ check:

make check

The `diff` shouldn't print any text. If it does, it means you have destroyed the original CAW.

```
/cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
rm caw caw.exe out.txt
rm: impossible de supprimer 'caw.exe': No such file or directory
rm: impossible de supprimer 'out.txt': No such file or directory
make: *** [Makefile:7: clean] Error 1

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ ls
caw.cc  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ make
g++ -static *.cc -o caw

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ ls
caw.cc  caw.exe  caw.h  golden_in.txt  golden_out.txt  main.cc  Makefile

bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$ make check
caw golden_in.txt out.txt
diff out.txt golden_out.txt

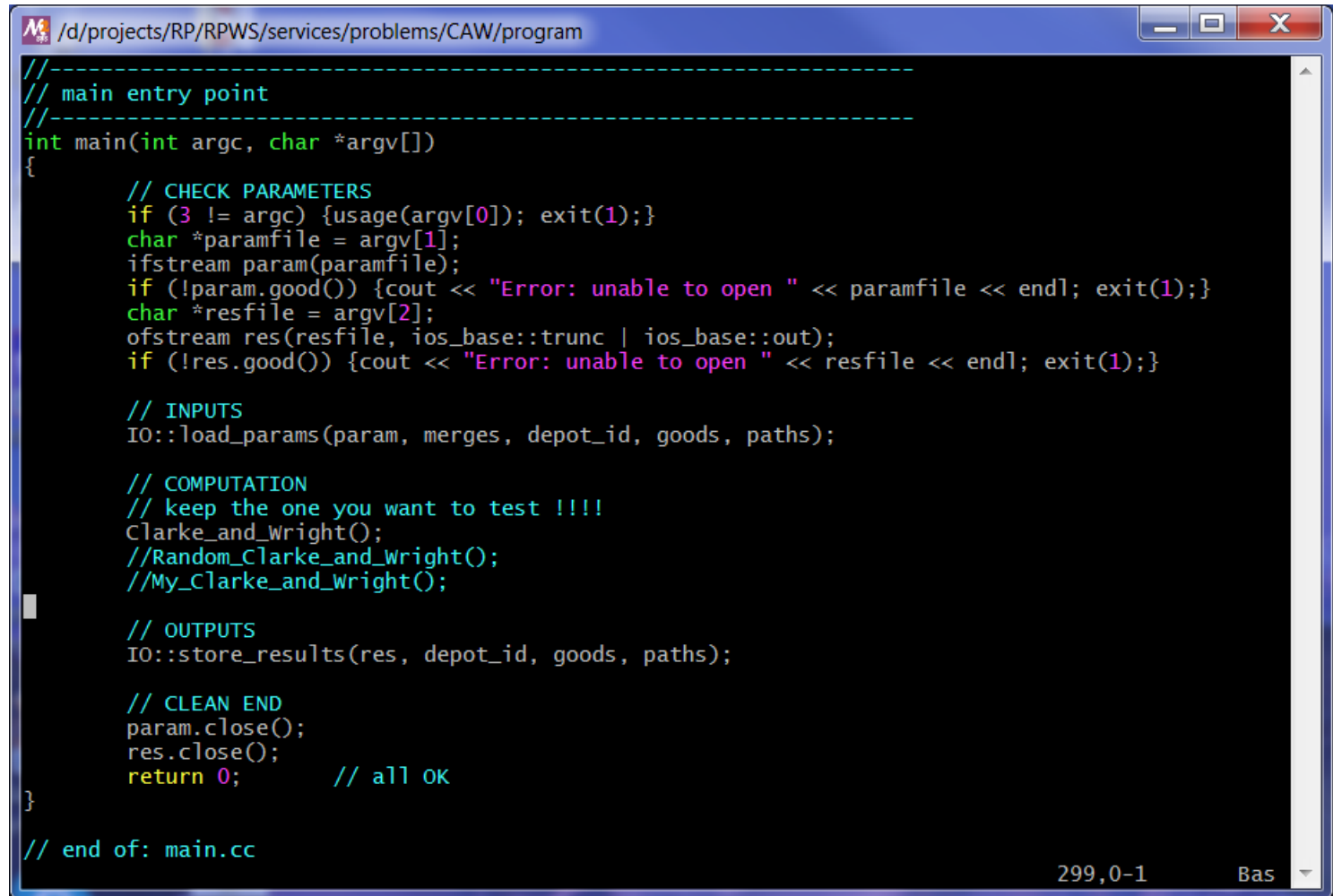
bomel@UBS-19760 /cygdrive/d/projects/RP/RPWS/services/problems/CAW/program
$
```

Clarke and Wright, C++

Open `main.cc` with your preferred editor and go at the bottom of the file to see the `main()` entry point.

I used `vim`.

I hope the text is self-explanatory enough.



```
//-----  
// main entry point  
//-----  
int main(int argc, char *argv[])  
{  
    // CHECK PARAMETERS  
    if (3 != argc) {usage(argv[0]); exit(1);}  
    char *paramfile = argv[1];  
    ifstream param(paramfile);  
    if (!param.good()) {cout << "Error: unable to open " << paramfile << endl; exit(1);}  
    char *resfile = argv[2];  
    ofstream res(resfile, ios_base::trunc | ios_base::out);  
    if (!res.good()) {cout << "Error: unable to open " << resfile << endl; exit(1);}  
  
    // INPUTS  
    IO::load_params(param, merges, depot_id, goods, paths);  
  
    // COMPUTATION  
    // keep the one you want to test !!!!  
    Clarke_and_Wright();  
    //Random_Clarke_and_Wright();  
    //My_Clarke_and_Wright();  
  
    // OUTPUTS  
    IO::store_results(res, depot_id, goods, paths);  
  
    // CLEAN END  
    param.close();  
    res.close();  
    return 0;        // all OK  
}  
  
// end of: main.cc
```

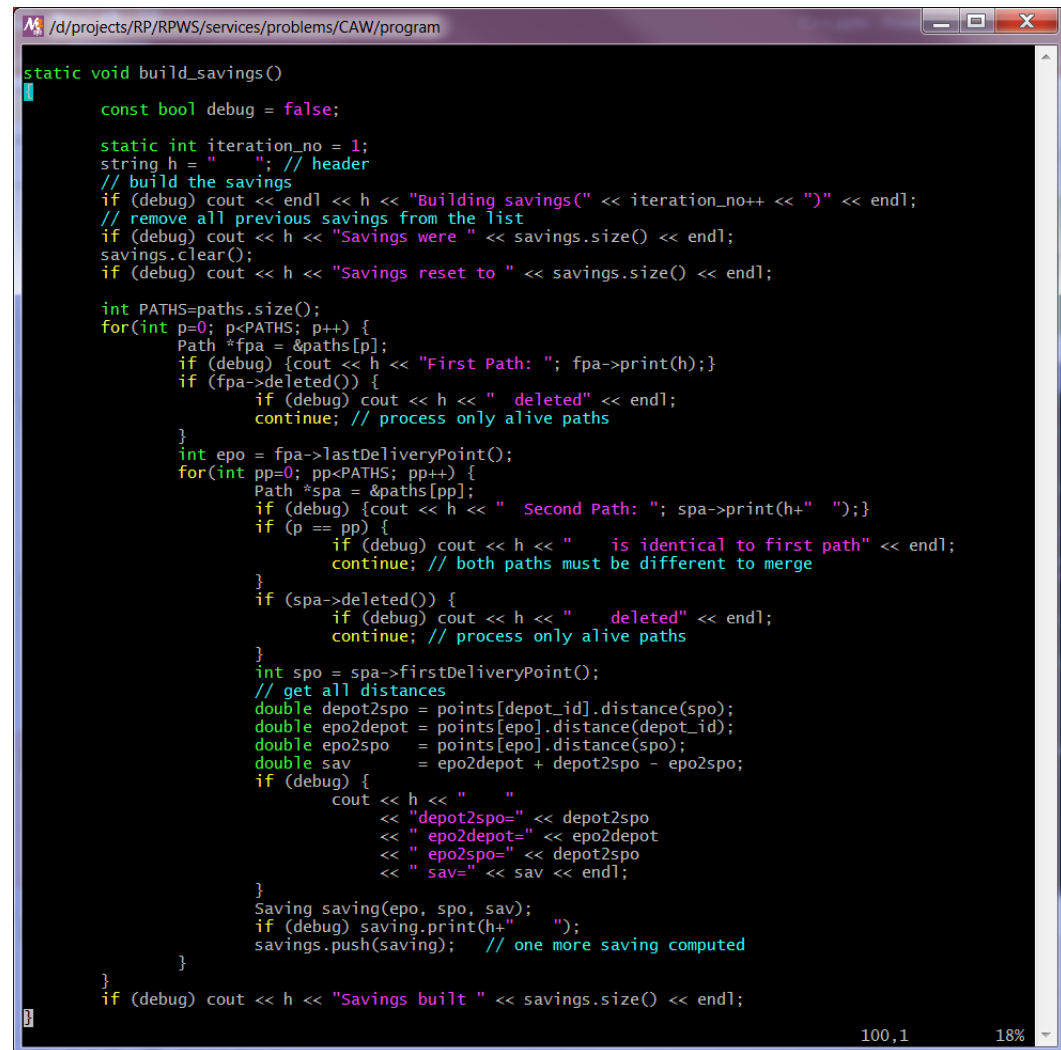
299,0-1 Bas

Clarke and Wright, C++

Go at the top of the file to see the `build_savings()` function.

You won't have to modify it.

But you could, to implement another search for possible merges.

A screenshot of a C++ code editor window. The title bar shows the file path: /d/projects/RP/RPWS/services/problems/CAW/program. The code is for a function named build_savings(). It starts with a static void declaration. Inside, there's a const bool debug = false; and a static int iteration_no = 1;. A string h is initialized with a space and a tab, labeled as a header. The function then builds savings by iterating through paths. It uses a for loop for p from 0 to PATHS-1. For each path, it checks if it's deleted. If not, it prints the first path and its delivery point. Then it iterates through other paths (pp) and checks if they are identical to the first path. If not identical and not deleted, it calculates the distance between the depot, the first path's delivery point, and the second path's delivery point, and pushes a saving into the savings list. The function ends with a cout statement showing the total savings built.

```
static void build_savings()
{
    const bool debug = false;

    static int iteration_no = 1;
    string h = "  "; // header
    // build the savings
    if (debug) cout << endl << h << "Building savings(" << iteration_no++ << ")" << endl;
    // remove all previous savings from the list
    if (debug) cout << h << "Savings were " << savings.size() << endl;
    savings.clear();
    if (debug) cout << h << "Savings reset to " << savings.size() << endl;

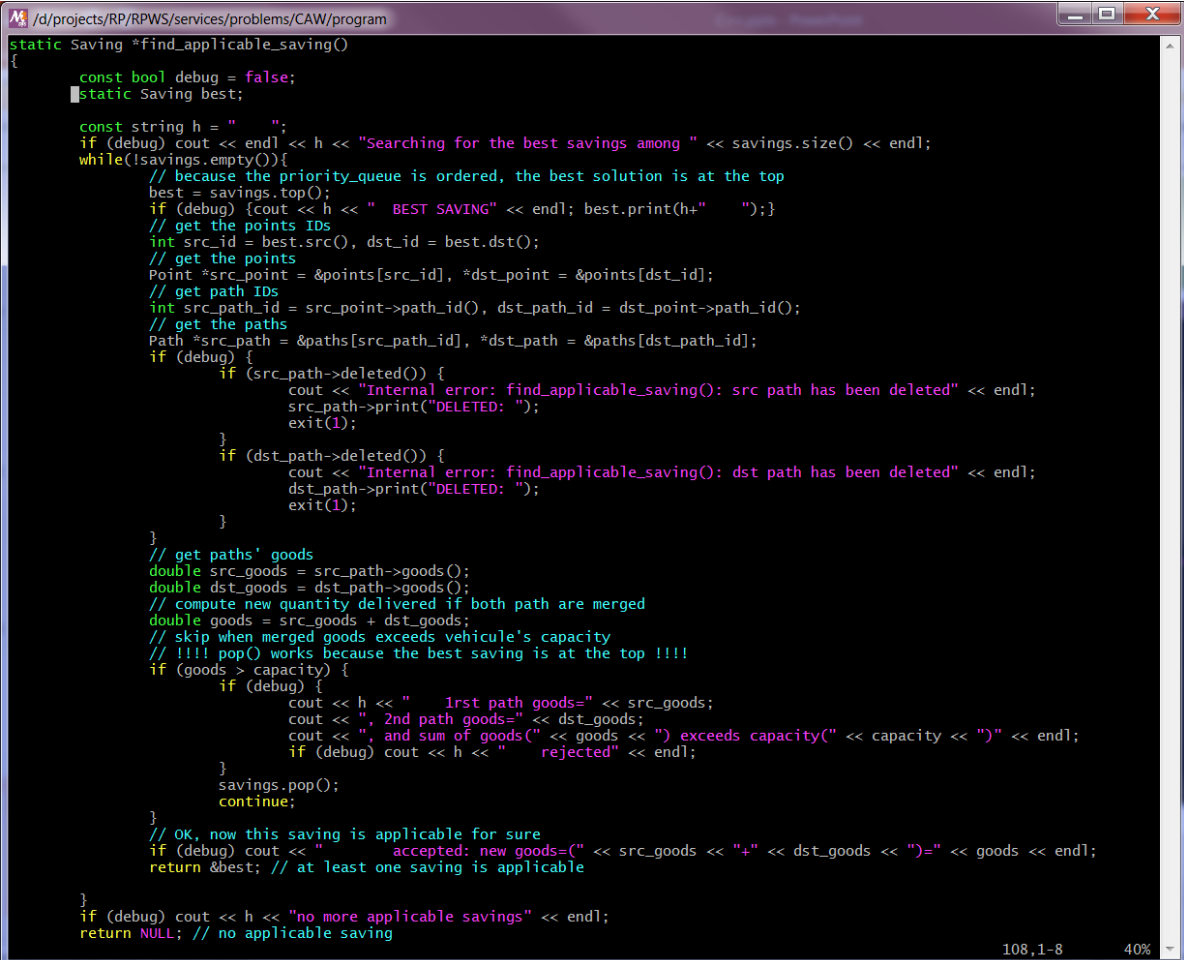
    int PATHS=paths.size();
    for(int p=0; p<PATHS; p++) {
        Path *fpa = &paths[p];
        if (debug) {cout << h << "First Path: "; fpa->print(h);}
        if (fpa->deleted()) {
            if (debug) cout << h << "  deleted" << endl;
            continue; // process only alive paths
        }
        int epo = fpa->lastDeliveryPoint();
        for(int pp=0; pp<PATHS; pp++) {
            Path *spa = &paths[pp];
            if (debug) {cout << h << "  Second Path: "; spa->print(h+" ");}
            if (p == pp) {
                if (debug) cout << h << "    is identical to first path" << endl;
                continue; // both paths must be different to merge
            }
            if (spa->deleted()) {
                if (debug) cout << h << "    deleted" << endl;
                continue; // process only alive paths
            }
            int spo = spa->firstDeliveryPoint();
            // get all distances
            double depot2spo = points[depot_id].distance(spo);
            double epo2depot = points[epo].distance(depot_id);
            double epo2spo = points[epo].distance(spo);
            double sav = epo2depot + depot2spo - epo2spo;
            if (debug) {
                cout << h << "      "
                    << "depot2spo=" << depot2spo
                    << "  epo2depot=" << epo2depot
                    << "  epo2spo=" << epo2spo
                    << "  sav=" << sav << endl;
            }
            Saving saving(epo, spo, sav);
            if (debug) saving.print(h+" ");
            savings.push(saving); // one more saving computed
        }
    }
    if (debug) cout << h << "Savings built " << savings.size() << endl;
}
```

Clarke and Wright, C++

Go further in the file to see the `find_applicable_savings()` function.

Also, you won't have to modify it.

But you could, to implement another Selection of the best merge.



```
/d/projects/RP/RPWS/services/problems/CAW/program
static Saving *find_applicable_saving()
{
    const bool debug = false;
    static Saving best;

    const string h = " ";
    if (debug) cout << endl << h << "Searching for the best savings among " << savings.size() << endl;
    while(!savings.empty()){
        // because the priority_queue is ordered, the best solution is at the top
        best = savings.top();
        if (debug) {cout << h << " BEST SAVING" << endl; best.print(h+ " ");}
        // get the points IDs
        int src_id = best.src(), dst_id = best.dst();
        // get the points
        Point *src_point = &points[src_id], *dst_point = &points[dst_id];
        // get path IDs
        int src_path_id = src_point->path_id(), dst_path_id = dst_point->path_id();
        // get the paths
        Path *src_path = &paths[src_path_id], *dst_path = &paths[dst_path_id];
        if (debug) {
            if (src_path->deleted()) {
                cout << "Internal error: find_applicable_saving(): src path has been deleted" << endl;
                src_path->print("DELETED: ");
                exit(1);
            }
            if (dst_path->deleted()) {
                cout << "Internal error: find_applicable_saving(): dst path has been deleted" << endl;
                dst_path->print("DELETED: ");
                exit(1);
            }
        }
        // get paths' goods
        double src_goods = src_path->goods();
        double dst_goods = dst_path->goods();
        // compute new quantity delivered if both path are merged
        double goods = src_goods + dst_goods;
        // skip when merged goods exceeds vehicule's capacity
        // !!!! pop() works because the best saving is at the top !!!!
        if (goods > capacity) {
            if (debug) {
                cout << h << " 1rst path goods=" << src_goods;
                cout << ", 2nd path goods=" << dst_goods;
                cout << ", and sum of goods(" << goods << ") exceeds capacity(" << capacity << ") << endl;
                if (debug) cout << h << " rejected" << endl;
            }
            savings.pop();
            continue;
        }
        // OK, now this saving is applicable for sure
        if (debug) cout << h << " accepted: new goods=(" << src_goods << "+" << dst_goods << ")=" << goods << endl;
        return &best; // at least one saving is applicable
    }
    if (debug) cout << h << "no more applicable savings" << endl;
    return NULL; // no applicable saving
}
```

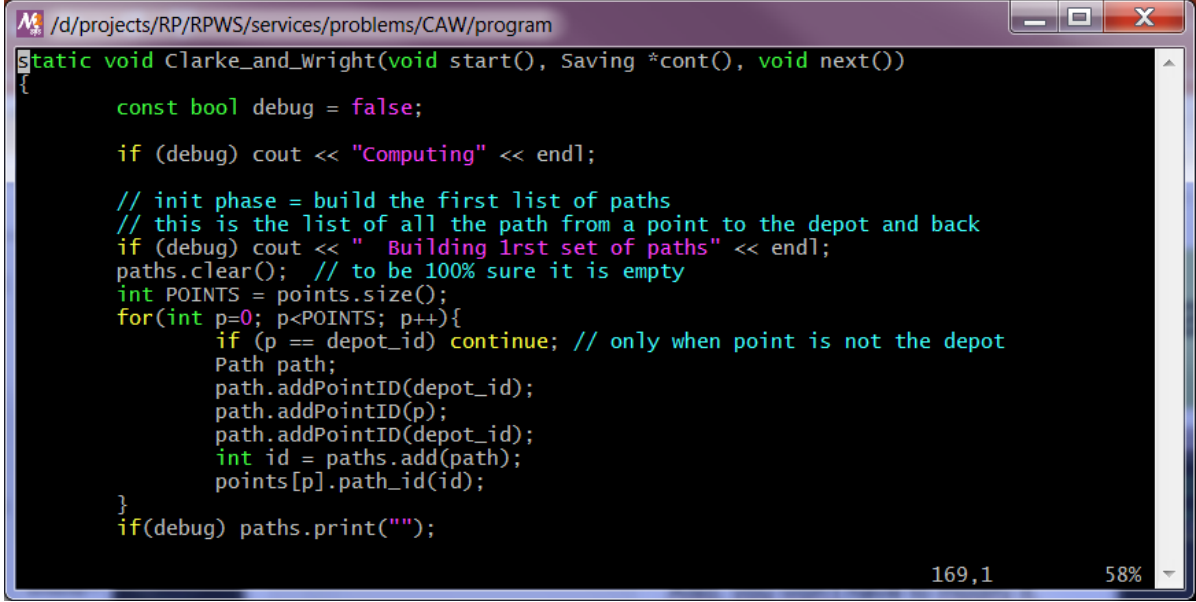
Clarke and Wright, C++

Go further in the file to see the `Clarke_and_Wright()` function.

This is the algorithm !

Its first part builds the set of all paths from the depot to each customer.

That's about 15 lines of code, debug included.

A screenshot of a C++ code editor window. The title bar shows the file path: /d/projects/RP/RPWS/services/problems/CAW/program. The code is for a static void function named Clarke_and_Wright, which takes void start(), Saving *cont(), and void next() as parameters. The code is color-coded: keywords are blue, comments are green, and identifiers are black. It includes a debug flag, a cout statement for "Computing", a comment about the init phase, another cout for "Building 1st set of paths", a clear() call on a paths vector, a loop over points, an if-statement to skip the depot, path construction with addPointID, adding the path to the paths vector, and updating the point's path_id. The code ends with a print statement if debug is enabled. The bottom right corner of the editor shows line 169, column 1 and a zoom level of 58%.

```
/d/projects/RP/RPWS/services/problems/CAW/program
static void Clarke_and_Wright(void start(), Saving *cont(), void next())
{
    const bool debug = false;

    if (debug) cout << "Computing" << endl;

    // init phase = build the first list of paths
    // this is the list of all the path from a point to the depot and back
    if (debug) cout << " Building 1st set of paths" << endl;
    paths.clear(); // to be 100% sure it is empty
    int POINTS = points.size();
    for(int p=0; p<POINTS; p++){
        if (p == depot_id) continue; // only when point is not the depot
        Path path;
        path.addPointID(depot_id);
        path.addPointID(p);
        path.addPointID(depot_id);
        int id = paths.add(path);
        points[p].path_id(id);
    }
    if(debug) paths.print("");
}
```

Clarke and Wright, C++

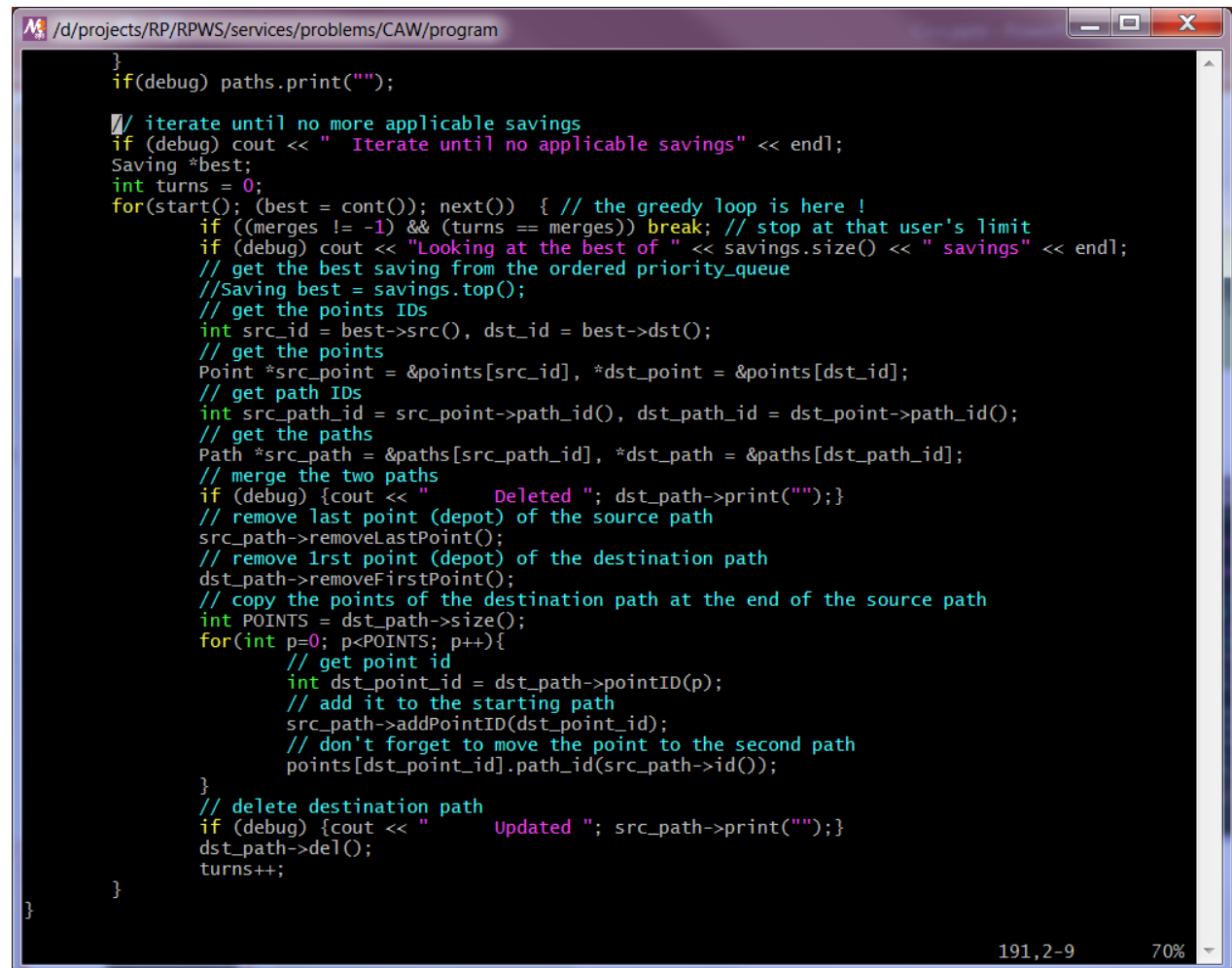
Go further in the file to see the `Clarke_and_Wright()` function.

This is the algorithm !

Its second part iteratively builds all the possible merges and selects the best one, if any.

The end of the algorithm occurs when there are no more applicable savings.

That's about 20 lines of code, debug included.



```

/d/projects/RP/RPWS/services/problems/CAW/program
}
if(debug) paths.print("");

// iterate until no more applicable savings
if (debug) cout << " Iterate until no applicable savings" << endl;
Saving *best;
int turns = 0;
for(start(); (best = cont()); next()) { // the greedy loop is here !
    if ((merges != -1) && (turns == merges)) break; // stop at that user's limit
    if (debug) cout << "Looking at the best of " << savings.size() << " savings" << endl;
    // get the best saving from the ordered priority_queue
    // Saving best = savings.top();
    // get the points IDs
    int src_id = best->src(), dst_id = best->dst();
    // get the points
    Point *src_point = &points[src_id], *dst_point = &points[dst_id];
    // get path IDs
    int src_path_id = src_point->path_id(), dst_path_id = dst_point->path_id();
    // get the paths
    Path *src_path = &paths[src_path_id], *dst_path = &paths[dst_path_id];
    // merge the two paths
    if (debug) {cout << " Deleted "; dst_path->print("");}
    // remove last point (depot) of the source path
    src_path->removeLastPoint();
    // remove 1st point (depot) of the destination path
    dst_path->removeFirstPoint();
    // copy the points of the destination path at the end of the source path
    int POINTS = dst_path->size();
    for(int p=0; p<POINTS; p++){
        // get point id
        int dst_point_id = dst_path->pointID(p);
        // add it to the starting path
        src_path->addPointID(dst_point_id);
        // don't forget to move the point to the second path
        points[dst_point_id].path_id(src_path->id());
    }
    // delete destination path
    if (debug) {cout << " Updated "; src_path->print("");}
    dst_path->del();
    turns++;
}
}

```

Clarke and Wright, C++

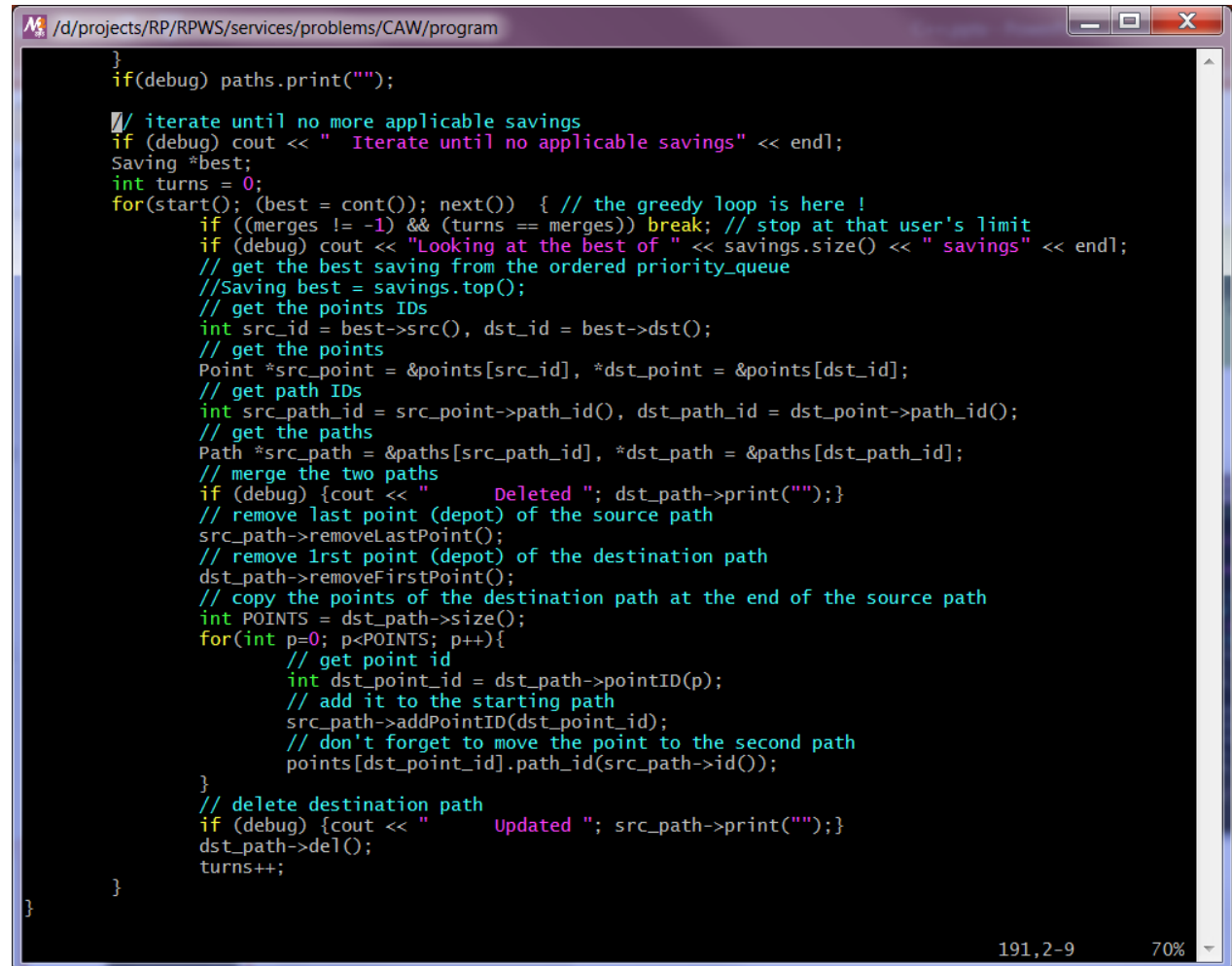
Shall you modify it ?

Even if this is only about 35 lines of C++ code, no you won't modify it today.

(some of you should feel more relaxed ...)

I've prepared something much simpler for you, because we do not have so much time.

You will do complex coding later at your office, in a calm and quiet context.



```
/d/projects/RP/RPWS/services/problems/CAW/program
}
if(debug) paths.print("");

// iterate until no more applicable savings
if (debug) cout << " Iterate until no applicable savings" << endl;
Saving *best;
int turns = 0;
for(start(); (best = cont()); next()) { // the greedy loop is here !
    if ((merges != -1) && (turns == merges)) break; // stop at that user's limit
    if (debug) cout << "Looking at the best of " << savings.size() << " savings" << endl;
    // get the best saving from the ordered priority_queue
    // Saving best = savings.top();
    // get the points IDs
    int src_id = best->src(), dst_id = best->dst();
    // get the points
    Point *src_point = &points[src_id], *dst_point = &points[dst_id];
    // get path IDs
    int src_path_id = src_point->path_id(), dst_path_id = dst_point->path_id();
    // get the paths
    Path *src_path = &paths[src_path_id], *dst_path = &paths[dst_path_id];
    // merge the two paths
    if (debug) {cout << " Deleted "; dst_path->print("");}
    // remove last point (depot) of the source path
    src_path->removeLastPoint();
    // remove 1st point (depot) of the destination path
    dst_path->removeFirstPoint();
    // copy the points of the destination path at the end of the source path
    int POINTS = dst_path->size();
    for(int p=0; p<POINTS; p++){
        // get point id
        int dst_point_id = dst_path->pointID(p);
        // add it to the starting path
        src_path->addPointID(dst_point_id);
        // don't forget to move the point to the second path
        points[dst_point_id].path_id(src_path->id());
    }
    // delete destination path
    if (debug) {cout << " Updated "; src_path->print("");}
    dst_path->del();
    turns++;
}
}
```

191,2-9 70%

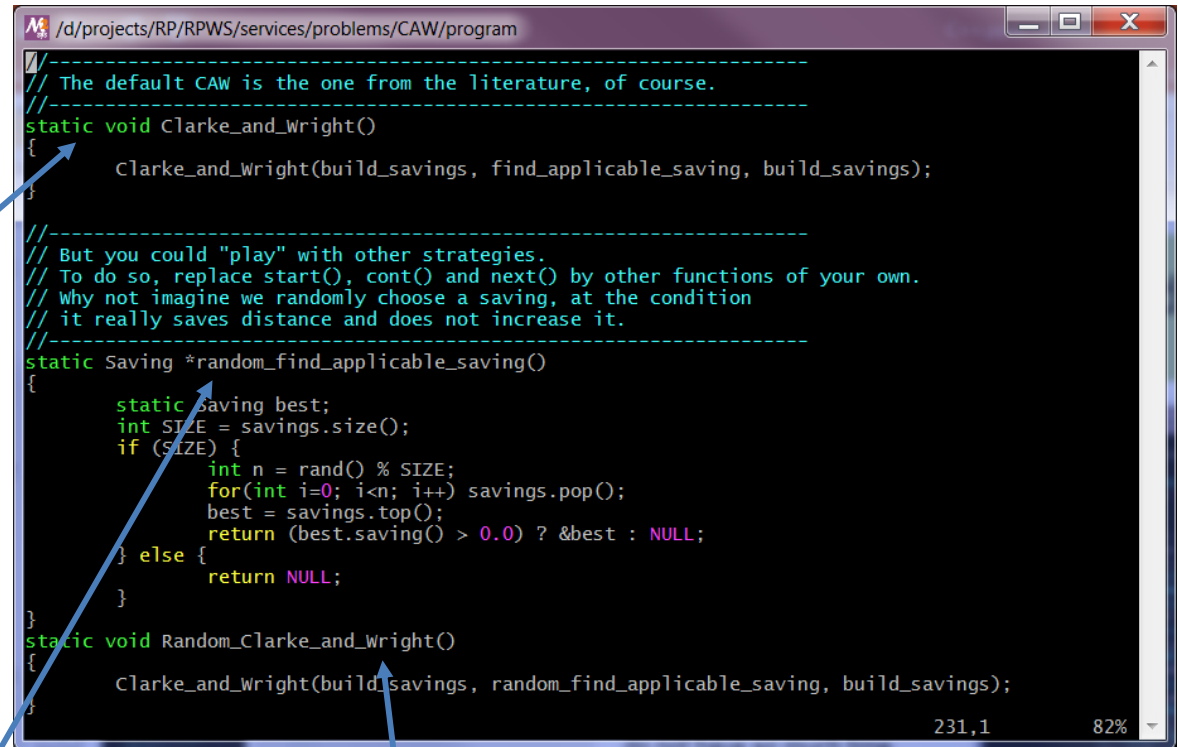
Clarke and Wright, C++

Go further in the file, until you find this.

The default `Clarke_and_Wright()` is the execution of the algorithm with the `build_savings()` and `find_applicable_saving()` functions we have looked at.

This is also what you have tested with the web-browser until now.

But you can execute the algorithm with other « saving-functions ». Here I have developed a new `random_find_applicable_saving()` which selects « randomly » a merge among the list of the possible merges. Of course it will not give the optimum, but it shows you how to modify the strategies with the `random_Clarke_and_Wright()`.



```
// /d/projects/RP/RPWS/services/problems/CAW/program
//-----
// The default CAW is the one from the literature, of course.
//-----
static void Clarke_and_Wright()
{
    Clarke_and_Wright(build_savings, find_applicable_saving, build_savings);
}

//-----
// But you could "play" with other strategies.
// To do so, replace start(), cont() and next() by other functions of your own.
// Why not imagine we randomly choose a saving, at the condition
// it really saves distance and does not increase it.
//-----
static Saving *random_find_applicable_saving()
{
    static Saving best;
    int SIZE = savings.size();
    if (SIZE) {
        int n = rand() % SIZE;
        for(int i=0; i<n; i++) savings.pop();
        best = savings.top();
        return (best.saving() > 0.0) ? &best : NULL;
    } else {
        return NULL;
    }
}

static void Random_Clarke_and_Wright()
{
    Clarke_and_Wright(build_savings, random_find_applicable_saving, build_savings);
}

231,1 82%
```

Clarke and Wright, C++

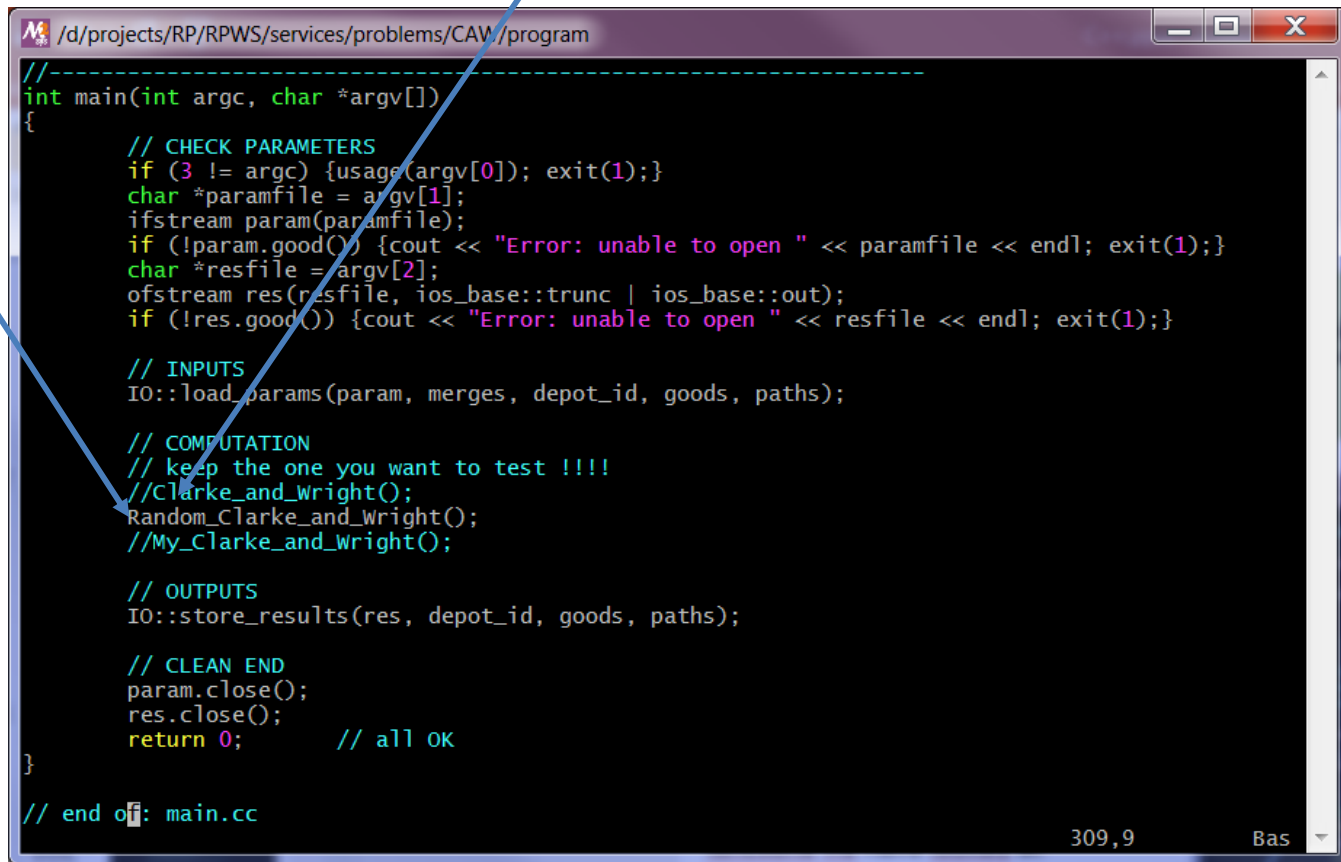
To test the random version
go back to the `main()` function, comment the « normal » call
and uncomment the call to the random version.

Easy isn't it ?

Now

save it
rebuild it
test it

with the web-browser



```
//-----  
int main(int argc, char *argv[])  
{  
    // CHECK PARAMETERS  
    if (3 != argc) {usage(argv[0]); exit(1);}  
    char *paramfile = argv[1];  
    ifstream param(paramfile);  
    if (!param.good()) {cout << "Error: unable to open " << paramfile << endl; exit(1);}  
    char *resfile = argv[2];  
    ofstream res(resfile, ios_base::trunc | ios_base::out);  
    if (!res.good()) {cout << "Error: unable to open " << resfile << endl; exit(1);}  
  
    // INPUTS  
    IO::load_params(param, merges, depot_id, goods, paths);  
  
    // COMPUTATION  
    // keep the one you want to test !!!!  
    // Clarke_and_Wright();  
    Random_Clarke_and_Wright();  
    // My_Clarke_and_Wright();  
  
    // OUTPUTS  
    IO::store_results(res, depot_id, goods, paths);  
  
    // CLEAN END  
    param.close();  
    res.close();  
    return 0;    // all OK  
}  
  
// end of: main.cc
```

Clarke and Wright, C++

Here is an example with 26 customers, dépôt # 1, capacity 10, with `random_Clarke_and_Wright()`

Routing Problems

Welcome to the Routing Problems website !

Problem: Clarke and Wright

Real instance

Data's instance

Distance matrix

Algorithms

select an algorithm:

name	time	rank	exact	exec.	info
CAW	<10s	1	no	RUN	?

Results:

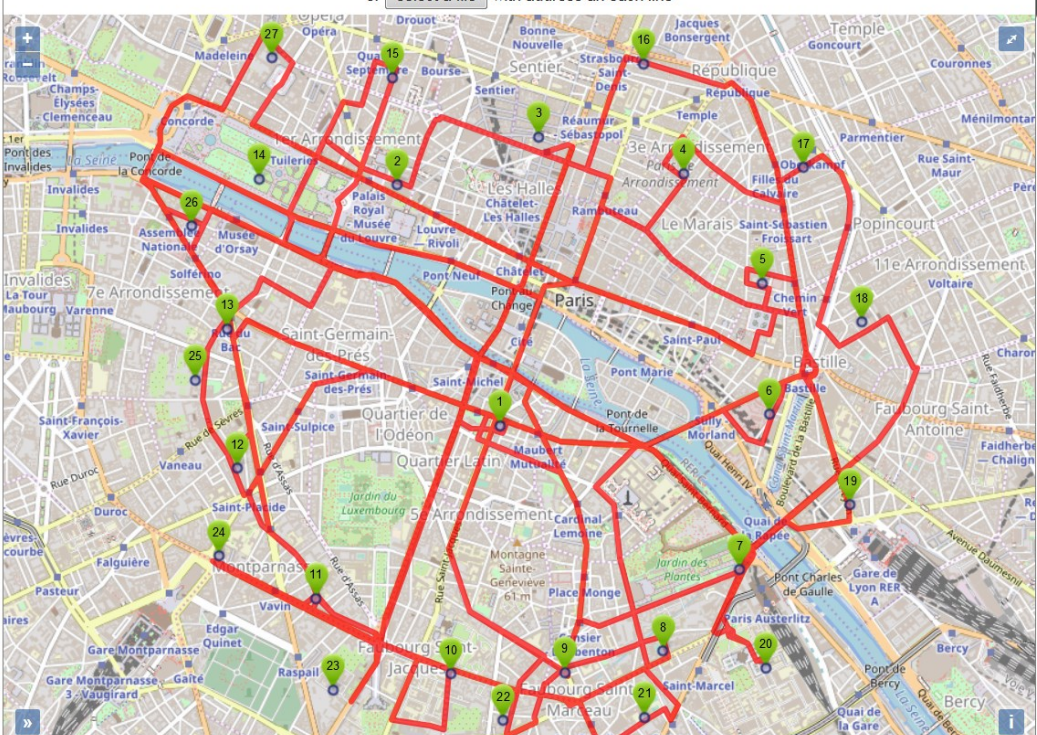
CAW	0:00:06			running...
CAW	0:00:00			

Solution from CAW (2)

trips: ALL 250 0 0 0.7

Properties:

runPos	1
cout	88341.5
duree	0
success	true
verif	0



Clarke and Wright, C++

While, with 26 customers, dépôt # 1, capacity 10, the original `Clarke_and_Wright()` gives

Routing Problems

Welcome to the Routing Problems website !

Problem: Clarke and Wright

Real instance

Data's instance

Distance matrix

Algorithms

select an algorithm:

name	time	rank	exact	exec.	info
CAW	<10s	1	no	RUN	?

Results:

CAW	0:00:06			running...
CAW	0:00:00			
CAW	0:00:00			

Solution from CAW (3)

trips: ALL 250 0 0 0.7

Properties:

runPos	2
cout	38870.6
duree	0
success	true
verif	0



Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

C++

The Clarke and Wright algorithm.

Modifying the CAW and testing it.

Adding your own strategy to the CAW.

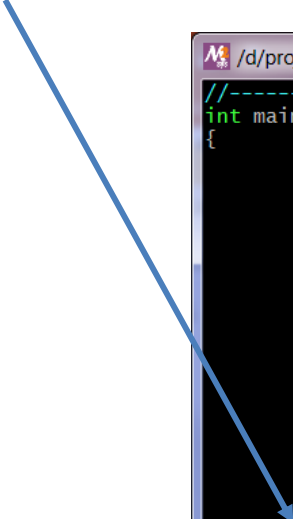
How to debug ?



Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

And now, because you must suffer a little bit, this is your turn to work !
You must add your `My_Clarke_and_Wright()` version of the algorithm.
I will stay close to you to help you until the end of the session.



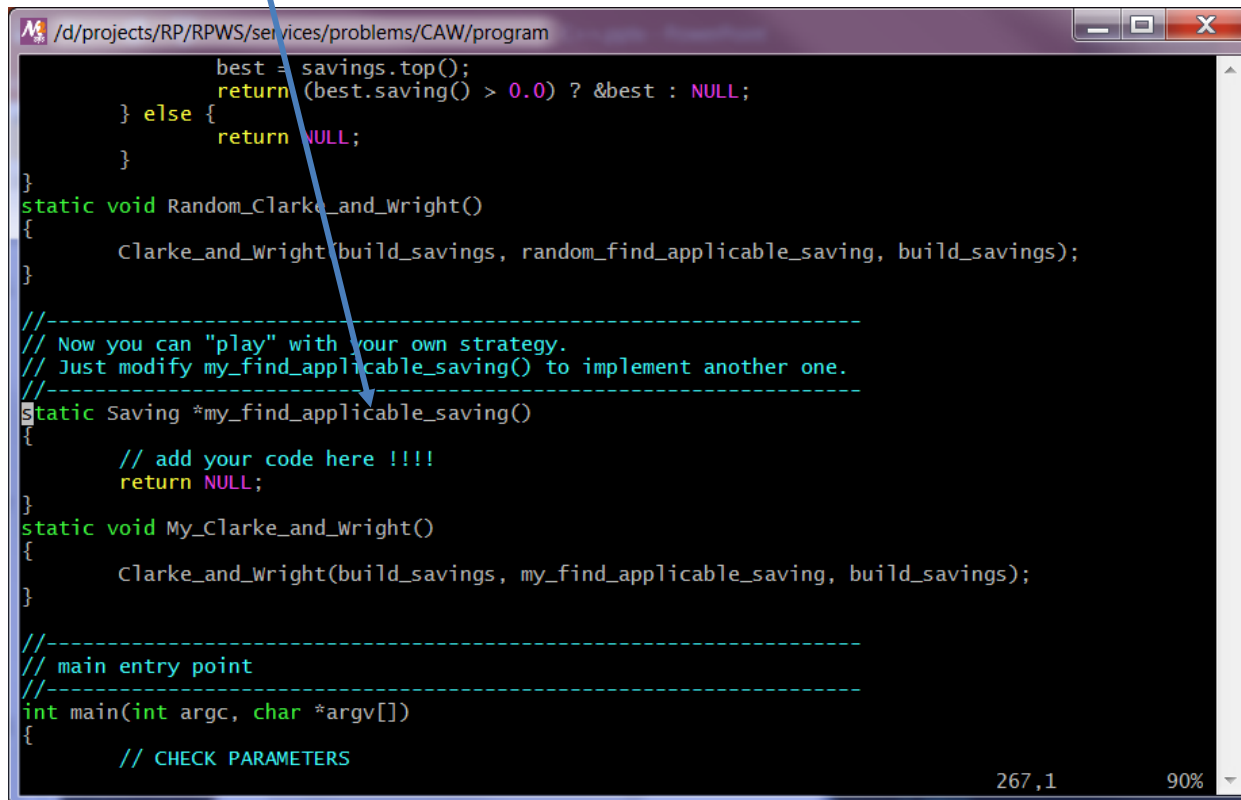
```
//-----  
int main(int argc, char *argv[])  
{  
    // CHECK PARAMETERS  
    if (3 != argc) {usage(argv[0]); exit(1);}  
    char *paramfile = argv[1];  
    ifstream param(paramfile);  
    if (!param.good()) {cout << "Error: unable to open " << paramfile << endl; exit(1);}  
    char *resfile = argv[2];  
    ofstream res(resfile, ios_base::trunc | ios_base::out);  
    if (!res.good()) {cout << "Error: unable to open " << resfile << endl; exit(1);}  
  
    // INPUTS  
    IO::load_params(param, merges, depot_id, goods, paths);  
  
    // COMPUTATION  
    // keep the one you want to test !!!!  
    //Clarke_and_Wright();  
    //Random_Clarke_and_Wright();  
    My_Clarke_and_Wright();  
  
    // OUTPUTS  
    IO::store_results(res, depot_id, goods, paths);  
  
    // CLEAN END  
    param.close();  
    res.close();  
    return 0;    // all OK  
}  
  
// end of: main.cc
```

298,2-9 Bas

Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

I suggest you to modify the strategy to differently choose the best saving.



```

/d/projects/RP/RPWS/services/problems/CAW/program
    best = savings.top();
    return (best.saving() > 0.0) ? &best : NULL;
} else {
    return NULL;
}
}
static void Random_Clarke_and_Wright()
{
    Clarke_and_Wright(build_savings, random_find_applicable_saving, build_savings);
}

//-----
// Now you can "play" with your own strategy.
// Just modify my_find_applicable_saving() to implement another one.
//-----
static Saving *my_find_applicable_saving()
{
    // add your code here !!!!
    return NULL;
}
static void My_Clarke_and_Wright()
{
    Clarke_and_Wright(build_savings, my_find_applicable_saving, build_savings);
}

//-----
// main entry point
//-----
int main(int argc, char *argv[])
{
    // CHECK PARAMETERS

```

Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

C++

The Clarke and Wright algorithm.
Modifying the CAW and testing it.
Adding your own strategy to the CAW.

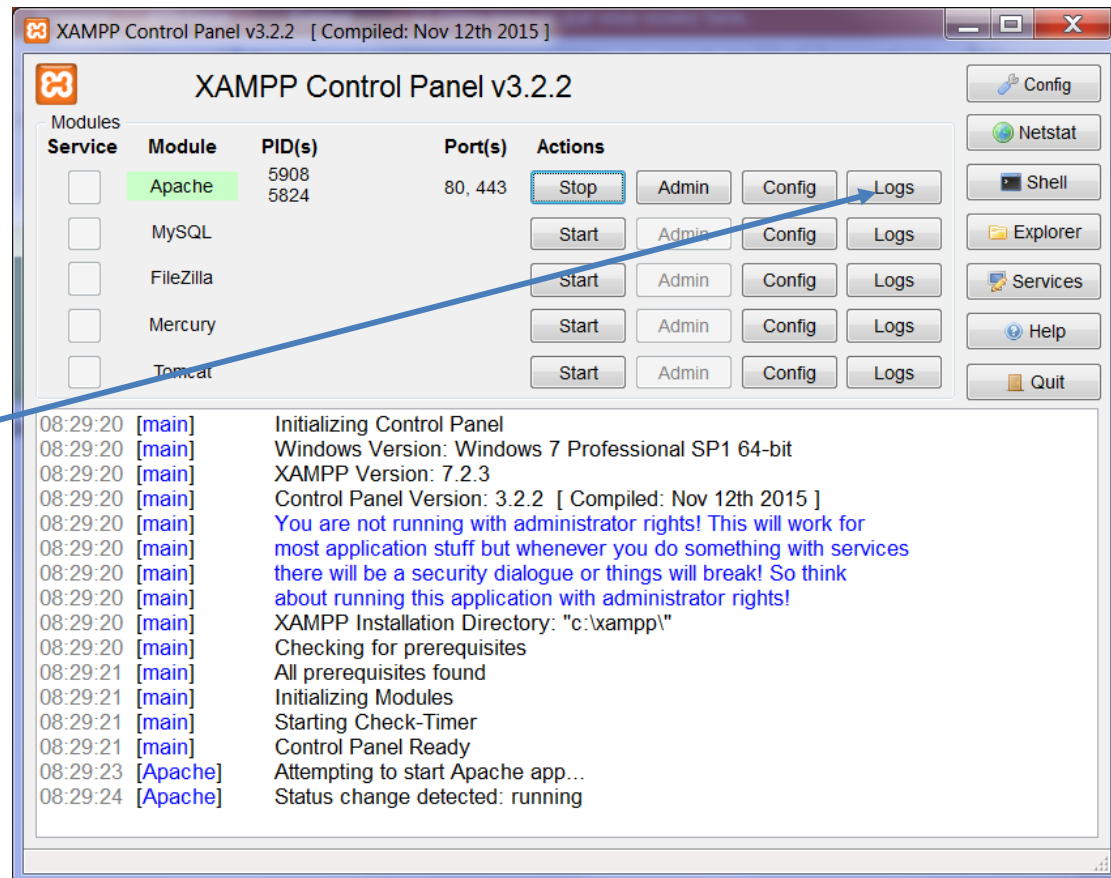
How to debug ?



Spring School on Integrated Operational Problems

May 14-16, 2018, Troyes, France

Of course, test your program before installing it as a web-service.
And don't forget to look at the error logs of the Apache server.



Click here and
select the log
you want to see

I'm sure you've made a good work !

Thank you for your kind attention.

See you soon at the second
session !

