Institut for Matematik og Datalogi
Syddansk Universitet

November 26, 2009

Peter Schneider-Kamp

Introduction to Computer Science
E09 – Week 11

Lecture: Monday, November 23

Jœrgen Bang-Jensen continued to lecture on combinatorial optimization based on the notes of Bjarne Toft.

Lecture: Monday, November 30, 12-14 (U140)

Marco Chiarandini will start to lecture on artificial intelligence based on Chapter 11.3.

Lecture, Wednesday, December 2, 14-16 (U28)

Marco Chiarandini will continue lecture on artificial intelligence based on Chapters 11.1, 11.2, 11.6, and 11.7.

Lecture: Monday, December 7, 12-14

Marco Chiarandini will start to lecture on artificial intelligence based on Chapters 11.4 and 11.5.

Discussion section: December 1, 10:15-12 (U37)

1. Exercise 36 on page 568 of the course book.
2. **Scheduling jobs on a single machine**.
   We are given a set J of jobs $\{1, \ldots, n\}$ to be processed on a single machine and for each job $j \in J$ a processing time p_j, a weight w_j stating its importance and a due date d_j.

   The task is to find a schedule, that is, an order for processing the jobs, that optimizes some given criterion.

(a) The completion time C_j of a job j is the time elapsed when a job is finished to be processed on the machine. Design an heuristic rule for the case where the criterion is minimizing the total weighted completion time, that is, $\sum_{j=1}^{n} w_j C_j$.

(b) The lateness L_j of a job j is the difference of the completion time from the due date, that is, $L_j = C_j - d_j$. Design an heuristic rule for the case where the criterion is minimizing the maximum lateness $L_{max} = \max_{j \in J} L_j$.

Compute your heuristics on the data of Figure 1.

Job	J_1	J_2	J_3	J_4	J_5	J_6
Processing Time	3	2	2	3	4	3
Weight	2	3	1	5	1	2
Due date	6	13	4	9	7	17

Job	J_3	J_1	J_5	J_4	J_1	J_6
C_j	2	5	9	12	14	17
L_j	-2	-1	2	3	8	0
$w_j C_j$	2	10	9	48	28	34

Figure 1: Data for the single machine scheduling problem and an example of computation of the parameters introduced for a sequence of jobs $\phi = J_3, J_1, J_5, J_4, J_1, J_6$.

3. **Packing bins.** Given is a finite set U of items, a size $s(u) \in Z^+$ for each $u \in U$, and a positive integer bin capacity B. The task is to find the minimal number of bins K for which there exits a partition of U into disjoint sets $U_1, U_2, \ldots, U_k$ and the sum of the sizes of the items in each U_i is B or less.

   (a) You may have already encountered this problem during the course in its online version. Here, you are asked to design heuristic rules to fit the items in the bins for the offline version of the problem, that is, when the information on all the items is available at once.

   (b) A similar and related problem is the **knapsack problem**. Here, we have only one bin available of size B and each item, beside its size, has associated a weight $w(u) \in Z^+$ indicating the profit it

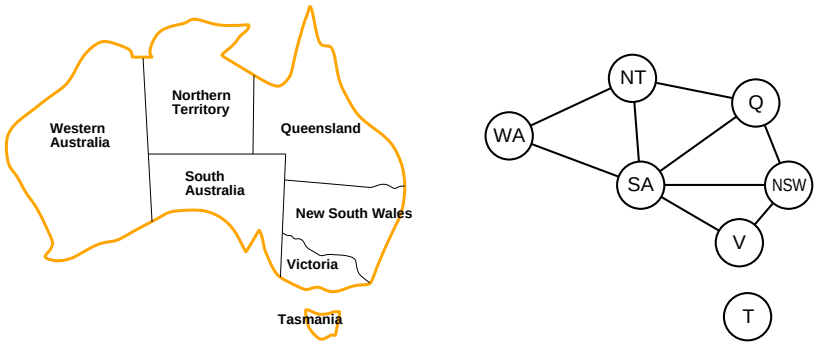

Figure 2: Coloring the states of Australia.

generates to place the item in the knapsack. The task is to decide which item to include in the knapsack in such a way that the capacity constraint is not violated and the total profit is maximized. Design heuristics rule to solve this problem.

4. **Coloring the vertices of a graph.** The graph (vertex) coloring problem consists in finding an assignment of colors to vertices of a graph in such a way that no adjacent vertices (that is, vertices connected by an edge) receive the same color and the number of colors used is the least possible.

   Design an heuristic for coloring graphs and try it on the graph of Figure 2. How many colors do you need to color a map such that adjacent states receives different colors?

Assignment due 14:15, December 10

Late assignments will not be accepted. Working together is not allowed. You may write this either in English or Danish. Write clearly if you do it by hand. Even better, use LaTeX.

The Travelling Salesman Problem

In the Traveling Salesman problem (TSP) we are given a set of points, each of them with coordinates x and y that identify their locations. The task is

to design a tour of *minimal length* that visits all the points exactly once and returns at the starting point.

The problem has many real life applications. It arises, of course, in transportation problems where by minimizing travel distances we save scarce resources and minimize environmental impact like CO_2 emissions. Postal deliveries is just one of such examples. Another application is when we are routing an autonomous guided vehicle (AGV) within some environment or sequencing drilling, welding or soldering tasks. In the industry we encounter at least two occurrences of this latter example: in a printed circuit board manufacturing process when masks for the production have to be drawn; and in a car manufacturing process when a robot has to weld certain parts. In both these cases an automatic system has to move through points in a plane, perform some operations at them and finally returning at the staring point. Other applications of the TSP are in scheduling jobs that are to be processed on a machine, in genome sequencing and in statistics for combinatorial data analysis, e.g., reordering rows and columns of data matrices to identify clusters.

In the case of n points, a brute force approach would enumerate all $(n-1)!$ possible different tours and choose the one of minimal length. For the applications above mentioned n might be very large and hence this solution approach is computationally prohibitive. However, the problem is known to be NP-complete, hence we do not know any algorithm that would solve this problem in a time bounded by a polynomial in n.

At the course web page you will find an `tar.gz` archive with some instances of the TSP and a Java code:

`http://imada.sdu.dk/~petersk/DM526/TSP.tar.gz`

The instances have varying size and geography. The small instance `ulysses16.tsp` provides the coordinates of the locations on the Mediterranean see visited by Ulysses in Homer's Odyssey (see Figure 3). The other instances have 1000 points. The instances `E1k.0 E1k.1 E1k.2` are points uniformly distributed in the plane, instances `C1k.0 C1k.1 C1k.2` are points located in clusters that are uniformly distributed in the plane.

The Java code implements some methods to read these instances, store the coordinate and distance of points, parse a command line, call to the program, compute a random tour and output its length and its sequence of points. It is also possible to plot the tour.

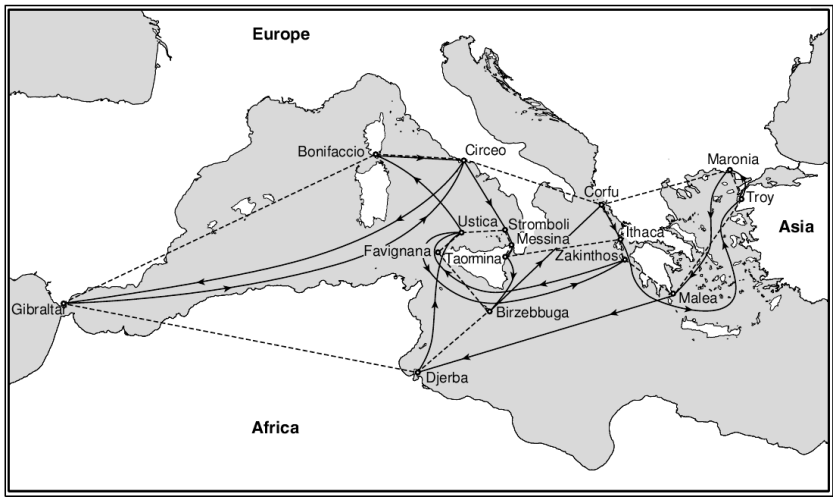

Figure 3: The legendary Ulysses was almost hopelessly lost on the two-dimensional sea surface and he did not choose the shortest way home. The solid line and arrows indicate the sequence in which Ulysses supposedly visited the 16 locations. The dashed lines indicate the optimal tour.

What you are required to do

Implement one heuristic algorithm to find the minimum cost tour. More specifically, you should **find a low cost solution for E1k.1 in the least amount of computation time**.

You should briefly describe in a short report (handwritten or type edited in Latex) your solution algorithm and report the results for some of the instances. For each instance you should report the length of the tour found by your algorithm and the time needed[1]. In addition, you must include a plot of your shortest tour found on the instance `E1k.1`. Write in the caption of this plot the length of the corresponding tour.

If you encounter problems in using the code made available and explained below, you can still pass the assignment by describing the heuristic you would implement in a report and showing its application carried out manually on the instance `ulysses16.tsp`. The description should be at a level of detail

[1]Hence, you should use the machines in IMADA terminal room. This will also guarantee that the code made available and everything explained in this text is installed and working correctly.

enough to allow a direct Java implementation.

You can write either in Danish or English. It makes no difference.

Possible ways to go

The code delivered will read a TSP instance and identify the points with the numbers $0, 1, 2, \ldots, n-1$. It will then create a symmetric matrix storing the distance between any pair of points (ignore entries in the diagonal). This is done in the class `Problem`. A solution to the TSP is a tour and it can be represented as a permutation of the points $0, 1, 2, \ldots, n-1$. This permutation is stored in an array and it represents the sequence with which points are to be visited[2]. This is stored in the class `Tour` representing the state of the search. The following approaches might be explored:

1. Generate a random tour by means of a random permutation. This is already implemented in the class `TourRandom` and can be used as a reference solution against which comparing your algorithms.
2. Use a greedy heuristic algorithm. For example, always moving to the nearest neighboring city.
3. Use an insertion heuristic algorithm that incrementally adds missing points to a growing tour inserting them in a clever position.
4. Restart the construction from different starting points chosen at random.

Implementation instructions

Download the archive TSP.tar.gz and uncompress it by typing:

```
~$ tar xzvf DM526-TSP.tar.gz
```

[2]Note, after visiting the point in the last position of the array we come back to the point in the first position, thus closing the trip

This will create a directory called `TSP-java-framework` and containing some Java files, a README file (with the same instructions reproduced here), a Makefile, two scripts `tsp` and `tsp-view` and a directory `data` containing the TSP instances. In addition, you will also have two directories `META-INF` and `lib` that you may ignore.

The Java files in `TSP-java-framework` implement two code frameworks: TSP and TSPView. The most important is the first, the second is for tour visualization and may be also ignored, if you decide to use `gnuplot` (see below).

The TSP framework comprises the following files: `RandomArray.java`, `Problem.java`, `Tour.java`, `TourSearch.java`, `TourRandom.java`, `TourYourAlg.java`, `TSP.java` to use in the implementation of the heuristics. It provides:

- utilities in `RandomArray.java` to generate random vectors and for parsing the command line in `lib/commons-cli-1.2.jar`
- a class `Problem` for reading the instance and storing data (`Problem.java`)
- a class `Tour` for maintaining the state of a solution (`Tour.java`)
- a class `TourSearch` for implementing algorithms to find tours (`TourSearch.java`) and two classes that extend this class, `TourRandom` and `TourYourAlg` (`TourRandom.java` and `TourYourAlg.java`);
- a class `TSP` to parse the command line, and call the procedures in `Problem`, `Tour` and `TourSearch`. (`TSP.java`)

In order to solve the assignment, **it is in principle enough to implement the function**:

```
public void construct(Tour t, Random r) {
          //TODO: Implement here your heuristic
    }
```

that you find in `TourYourAlg.java` together with some further pieces of information.

Note that internally, in the Java program, points are represented by numbers from 0 to $n-1$. Externally, (eg, in the instance file and in the solution file that is produced) they are represented with numbers from 1 to n.

TSPView is a small program to visualize the solutions found. You can call it from the TSP framework as you are constructing the tour to see how your technique progresses. The class `TourView` (`TourView.java`) provides the methods to make the plots while `TSPView.java` gives an example of how to call the methods.

To compile the programs type:

```
~$ make all
```

This will compile the Java files with

```
~$ javac -cp .:lib/commons-cli-1.2.jar *.java
```

(the `-cp` part is for importing the libraries needed by the command line parser) and create a `jar` archive with:

```
~$ jar cmf META-INF/manifest.mf tsp.jar *.class
```

Then to run the program call, as usual, `java TSP` or use the `tsp` wrapper:

```
~$ ./tsp -i data/E1k.1 -o E1k.1.sln -ch Random -t 10 -s 6
```

This will run the program with the Random heuristic and random seed 6. The time limit is not implemented because the heuristic is in any case very fast. If the `-ch` flag is omitted then by default the Random heuristic is used. When it finishes the program will output:

```
TIMELIMIT: 10
INSTANCE: data/E1k.1
SEED: 6
OUTPUTFILE: sol
HEURISTIC: Random
TOURLENGTH: 332717
TIME: 0.0020
```

where the relevant data are `TOURLENGTH` and `TIME` (in seconds). Moreover, it will print the solution in `E1k.1.sln`

```
~$ head E1k.1.sln
852 80464.0 9585.0
576 192626.0 548478.0
77 351091.0 714240.0
257 39478.0 623807.0
802 930104.0 252332.0
808 212948.0 691584.0
590 726971.0 805267.0
738 266706.0 438713.0
71 606244.0 240744.0
837 89413.0 532908.0
```

indicating that the tour will start at point 852, visits 576, etc. The third and second columns give the coordinates of the points. This information is useful for plotting the tour.

There are two ways you can plot a tour. You can use `TSPView` or `gnuplot`. TSPView is compiled with `make all` and can be invoked as follows:

```
~$ ./tsp-view <instance_file> <solution_file>
```

To use `gnuplot` you should first start the program from the shell typing `gnuplot` and then:

```
gnuplot> plot 'E1k.1' using 2:3 with linespoints
```

The advantage of using gnuplot is that you can print the plot in a postscript file and thus include it in a Latex document. To do this, after having made the plot as indicated above, type:

```
gnuplot> set term postscript     (will produce postscript output)
gnuplot> set output "printme.ps" (output to any filename you use)
gnuplot> replot                  (recreates plot, goes directly to file)
```

Then if you want a pdf you can run `ps2pdf printme.ps printme.pdf` (or similarly with `pstopdf`). Alternatively, if you want an image file:

```
gnuplot> set term png             (will produce .png output)
gnuplot> set output "printme.png" (output to any filename you use)
gnuplot> replot
```