Algorithms and Data Structures 3 (course 1DL481)
Uppsala University – Spring 2017
Assignment 1

Prepared by Pierre Flener, Di Yuan, and Gustav Björdal
— Deadline: 14:00 on Friday 3 February 2017 —

It is strongly recommended to read the Submission Instructions and Grading Rules at the end of this document before attempting to solve the following problems.

Problem 1: Stochastic Local Search (SLS)

An investment design \langle v, b, r \rangle, with \( v \geq 2 \) and \( b \geq r \geq 1 \), is a matrix of \( v \) rows and \( b \) columns of 0-1 integer values such that each row sums to \( r \), and the largest dot product between all pairs of rows is minimal. Equivalently, there are \( v \) subsets of size \( r \) within a given set of \( b \) elements, such that the largest intersection of any two of the \( v \) sets has minimal size.

For example, the following figure shows two \langle 10, 8, 3 \rangle investment designs, where grey cells represent value 1 and white cells represent value 0:

In these investment designs, there are dot products (intersection sizes) of 0 to 2, and their largest dot products are both 2, which is minimal: there exists no \langle 10, 8, 3 \rangle investment design where 1 is the largest dot product \[1\].

This is an abstract description of a problem that appears in finance (see \url{http://akas.imdb.com/title/tt1596363} The Big Short). In a typical investment design in finance, we have \( 4 \leq v \leq 25 \) and \( 250 \leq b \leq 500 \), with \( r \approx 100 \).

A lower bound on the number, \( \lambda \), of shared elements of any pair among \( v \) subsets of size \( r \) drawn from a given set of \( b \) elements is given in \[1\]:

\[
\lambda \geq \left\lceil \frac{\left\lfloor \frac{rv}{b} \right\rfloor^2 ((rv) \text{ mod } b) + \left\lfloor \frac{rv}{b} \right\rfloor^2 (b - ((rv) \text{ mod } b)) - rv}{v(v-1)} \right\rceil
\] (1)
For 20 of the following 21 instances, the lower bound (lb) on $\lambda$ is known to be feasible:

\[
\begin{array}{cccc|cccc|cccc}
 v & b & r & lb(\lambda) & v & b & r & lb(\lambda) & v & b & r & lb(\lambda) \\
10 & 360 & 120 & 32 & 9 & 70 & 35 & 16 & 19 & 19 & 9 & 4 \\
12 & 200 & 75 & 25 & 11 & 150 & 50 & 14 & 11 & 22 & 10 & 4 \\
9 & 300 & 100 & 25 & 6 & 60 & 30 & 12 & 9 & 36 & 12 & 3 \\
15 & 350 & 100 & 24 & 6 & 50 & 25 & 10 & 16 & 16 & 6 & 2 \\
10 & 325 & 100 & 24 & 10 & 100 & 30 & 7 & 15 & 21 & 7 & 2 \\
13 & 250 & 80 & 22 & 8 & 28 & 14 & 6 & 12 & 44 & 11 & 2 \\
10 & 350 & 100 & 22 & 10 & 37 & 14 & 5 & 10 & 30 & 9 & 2 \\
\end{array}
\]

For the instance $\langle 15, 21, 7 \rangle$ in the table, the lower bound 2 is infeasible and a best solution has $\lambda = 3$. For the instance $\langle 10, 8, 3 \rangle$ discussed above, the lower bound is 1, which is infeasible [1].

**Tasks:** You **must** follow the structure of the demo report!

a. Design and implement, in a programming language for which a compiler or interpreter is available on the Linux computers of the IT department, a stochastic local search algorithm, based on simulated annealing or tabu search, for computing or over-approximating investment designs.

An executable called `invDes` must read the space-separated problem parameters $v$, $b$, and $r$ from standard input, say `invDes 10 360 120`, and write to standard output a line with the space-separated values of $v$, $b$, $r$, $lb(\lambda)$, and $\lambda$, followed by one line per row of a $v \times b$ matrix representing the solution, the 0-1 cell values being space-separated. For example, the $\langle 10, 8, 3 \rangle$ investment design on the left of the previous page is represented by `http://user.it.uu.se/~pierref/courses/AD3/assignments/invDes-10-8-3.txt`.

You can pipe the output into the polynomial-time solution checker at `http://user.it.uu.se/~pierref/courses/AD3/assignments` (for Linux and macOS), which reads such a solution from standard input, in order to gain confidence in the correctness of your algorithm.

**Hints:** Satisfy the row sum (or subset size) constraint in the initial assignment and preserve the satisfaction of that constraint by every considered move: your algorithm will at any moment have an assignment satisfying all the constraints and can focus on trying to reduce the largest dot product (subset intersection size). Pay particular attention to designing data structures that enable the efficient probing of the moves. In stochastic local search, it often backfires to eliminate symmetric candidate solutions from the search space: do not investigate this.

b. Show, with plots or tables, how you have evaluated the performance of your algorithm and fine-tuned it via **at least two** algorithm parameter configurations, by making experiments on the 21 instances in the table above, proceeding by increasing value of $lb(\lambda)$; craft smaller instances if initially needed.

Use a time-out of at least 300 seconds per run and report the **median** (not: average) performance (runtime, steps, and achieved largest dot product $\lambda$) over at least 5 independent runs per instance, as a stochastic local search algorithm must be randomised by definition; a precision of one decimal place suffices here.

It is important that you automate the running of the experiments via a script. The sharing of scripts via the Discussion forum on the Student Portal is allowed and even encouraged.
The larger instances in the table above may be difficult for your algorithm: you are not expected to reach or get very close to all their optimal values of $\lambda$ discussed above.

c. Explain how you use the bound in (1) and the impact compared to not using it.

d. Outline (in plain English or in high-level pseudocode, but without an implementation) an exact algorithm (performing brute-force search, if you want) for the investment design problem. Express the size of the search space of your exact algorithm in terms of the problem parameters $v$, $b$, and $r$. For each instance in the table above that your stochastic local search algorithm solves to proven optimality before timing out, state how many candidate solutions your exact algorithm would have to examine per second in order to match the runtime performance of the seemingly best algorithm parameter setting, according to Task b of your stochastic local search algorithm.

Solo teams may omit reporting the algorithm parameter fine-tuning part of Task b (but should still show the performance on one algorithm parameter setting) as well as Task d.

References


Problem 2: Mixed Integer Programming (MIP)

**Background and Motivation:** The following optimisation problem is motivated by locating emergency-response service stations, which have fire trucks, for example. In the planning process, the service region is divided into zones. Each zone represents a candidate location of an emergency-response service station. Each zone has a demand: the expected frequency of service request, based on historical data and estimation. When the emergency-response agency receives a call, originating from some zone, the operator usually dispatches a vehicle from the closest (measured in travel time) service station, which may or may not be in the same zone.

For a given planning solution, the service quality for a zone is primarily the distance (measured in travel time) from the closest zone in which the service station is located, or, equivalently, the travel time of the closest vehicle. To account for the fact that one or even multiple vehicles of the closest service station may become occupied for serving other requests, we consider the average travel time of the three closest vehicles, which may be located at different service stations. Furthermore, this average time is multiplied with the demand, so zones with high demand will receive more weight in the optimisation process.

**Problem Specification:** The service station location problem has the following inputs:

- $z$ is the number of zones in the service region;
- $s$ is the number of service stations to be located within the zones;
- $v$ is the number of vehicles hosted at each service station;
- $\text{Demand}[i]$ is the demand of zone $i$, with $i \in \{1, \ldots, z\}$;
• $Time[i,j]$ is the travel time from zone $i$ to zone $j$; note that the matrix need not be symmetric (hence pay attention to get the indices right); the travel time within a zone is typically a small value, though not zero.

The optimisation decision consists in selecting exactly $s$ out of the $z$ zones in order to locate service stations. The objective function is the average travel time of the three closest vehicles of each zone, multiplied by the demand of the zone, and then summed over all the zones.

**Data Format:** The problem parameters in the AMPL data files at [http://user.it.uu.se/~pierref/courses/AD3/assignments/servStatLoc-data](http://user.it.uu.se/~pierref/courses/AD3/assignments/servStatLoc-data) are illustrated below, for $z = 10$ zones, $s = 2$ service stations, and $v = 2$ vehicles per service station:

```plaintext
param z := 10;
param s := 2;
param v := 2;
param Demand :=
  1  0.000803429155663251
  2  0.000951406002100867
  ...
  10 0.000597429377117032;
param Time :=
  1  1 0.0010
  1  2 1.6420
  ...
  1 10 4.0630
  2  1 1.6420
  ...
  10 10 0.0010;
```

where:

• the parameter Demand, indexed over the zones, gives the $z$ demand values $Demand[i]$;
• the parameter Time has $z^2$ elements specifying the travel times $Time[i,j]$; for example, the travel time from zone $i = 1$ to zone $j = 10$ is 4.063 in the data above.

In an AMPL model, use the following declarations in order to match up with the data files:

```plaintext
param z;
param s;
param v;
set Zones := 1..z;
param Demand {Zones} >= 0;
param Time {Zones,Zones} >= 0;
```

**Tasks:** You must follow the structure of the demo report!

a. Model the service station location problem as a mixed integer linear program. Explain the meaning of the decision variables that you use. Do not worry about symmetric candidate solutions in the search space: MIP solvers automatically exploit symmetries.

b. Implement the resulting model into an AMPL model called servStatLoc.mod. Indicate the MIP solver you have chosen for your experiments.
c. Solve the problem for \( z = 10, s \in \{2, 3, 4\}, v = 2 \) using data files `location-010-0s-2.dat`, where only parameter \( s \) varies. For your convenience, the optimum objective value for \( s = 2 \) is 0.008740338682. When \( s \) increases, one has a higher budget for service stations and one has more vehicles overall: discuss how much the objective value improves.

d. Solve the problem for \( z = 20, s \in \{2, 3, 4, 5, 6\}, v = 2 \) using files `location-020-0s-2.dat`, where only parameter \( s \) varies. For your convenience, the optimum objective value for \( s = 2 \) is 0.02324626135. Do you obtain much improvement when \( s \) grows beyond 4?

e. Solve the problem for \( z = 40, s = 5, v = 2 \) using data file `location-040-05-2.dat`.

f. Solve the problem for \( z = 80, s = 8, v = 2 \) using data file `location-080-08-2.dat`. In data file `location-080-16-1.dat`, only the parameters \( s \) and \( v \) are changed, to \( s = 16 \) and \( v = 1 \): the number of vehicles remains 16, but there are \( s = 16 \) service stations with \( v = 1 \) vehicle each, instead of \( s = 8 \) service stations with \( v = 2 \) vehicles each. Compare the optimum objective values: which one is better, and why?

g. Try and solve the problem for \( z = 120, s = 10, v = 2 \) using file `location-120-10-2.dat`. If your model times out, then propose an algorithm for delivering a not necessarily optimal solution, such that the algorithm is expected to have reasonable running time. Specify the steps of the algorithm. You do not need to implement and run the algorithm.

h. Express the size of the search space of a brute-force search algorithm in terms of the problem parameters \( z, s, \) and \( v \). For each instance mentioned above that your chosen MIP solver has solved to proven optimality before timing out, state how many candidate solutions a brute-force search algorithm would have to examine per second in order to match the reported runtime performance of the chosen MIP solver on your AMPL model.

Solo teams may omit Tasks g and h.

Use a time-out of at least 300 seconds per run and report the performance (runtime, objective value, and optimality gap) over a single run per instance, as the recommended solvers are deterministic by default; a precision of two decimal places suffices here for the runtime and optimality gap, but the objective value should be given in full precision.

**Submission Instructions**

All task answers, other than source code, must be in a single report in PDF format; all other formats are rejected. Furthermore:

- Identify the team members and state the team number inside the report.
- Certify that your report and all its uploaded attachments were produced solely by your team, except where explicitly stated otherwise and clearly referenced, that each teammate can individually explain any part starting from the moment of submitting your report, and that your report and attachments are not and will not be freely accessible on a public repository.
- State the problem number and task identifier for each answer in the report.
- Take the instructions of the demo report at [http://user.it.uu.se/~pierref/courses/AD3/demoReport](http://user.it.uu.se/~pierref/courses/AD3/demoReport) as a strict guideline for document structure and content, as well as an indication of its expected quality.
• Write clear task answers, source code, and comments.
• Justify all task answers, except where explicitly not required.
• State any assumptions you make that are not in this document.
• Thoroughly proofread, spellcheck, and grammar-check your report.
• Upload all source code required to run your experiments, and include running instructions.
• Match exactly the uppercase, lowercase, and layout conventions of any filenames and I/O texts imposed by the tasks, as we may process your source code automatically.
• Write a paragraph, which will not be graded, describing your experience with this assignment. Which tasks were too difficult or too easy? Which tasks were interesting or boring? This will help us improve the course in the coming years.

Only one of the teammates submits the solution files (one PDF report with answers to all the tasks, plus all source code files), without folder structure and without compression, via the Student Portal, whose clock may differ from yours, by the given hard deadline.

Grading Rules

For each problem: If the requested source code exists, and runs without runtime errors on the Linux/Unix computers of the IT department under the compiler, interpreter, or solver you indicate, and computes (near-)optimal solutions to some of our chosen instance data in reasonable time on our chosen hardware, and has all the requested task answers in the report, then you get at least 1 point (of 5), otherwise you get 0 points. Furthermore:

• If your code passes most of our grading tests, then you get 4 or 5 points, depending also on the quality of the report part for this problem; you are not invited to the grading session for this problem.

• If your code fails many of our grading tests, then you get an initial mark of 1 or 2 points, depending also on the quality of the report part for this problem; you are invited to the grading session for this problem, where you can try and increase your initial mark by 1 point into your final mark.

However, if the assistant figures out a minor fix that is needed to make your source code run as per your and our instructions, then, instead of giving 0 points up front, the assistant may deduct 1 point at his discretion.

Considering that there are two help sessions for each assignment, you must get minimum 3 points (of 10) on each assignment, including minimum 1 point (of 5) on each problem, until the end of its grading session, and minimum 10 points (of 20) over both assignments; otherwise you fail the Assignments part of the course.