Algorithms and Data Structures III (course 1DL481)
Uppsala University – Spring 2026
Report for Assignment 1 by Team t

Clara CLÄVER and Whiz KIDD

17th December 2025

Problem 1: Mixed Integer Programming (MIP)

Task A. What are the variables, their meanings, their constraints, and the objective function? For example, for the investment design problem, one might write: "Let variable m_{ij} take value 1 if basket i invests in credit j, and value 0 otherwise, with $i \in 1\,..\,v$ and $j \in 1\,..\,b$. The constraint that each row must sum up to r can then be linearly modelled as

$$\forall i \in 1\,..\,v : \sum_{j \in 1..b} m_{ij} = r$$

Etc."

(We are aware that we may lose 1 point if the model described above is different from what is actually implemented in Task B and evaluated in Tasks C to H.)

Task B. Our model `servStatLoc.mod` is uploaded with this report (but not listed inside it): we checked that its constraints and objective function are linear (and we are aware that 4 points will otherwise be deducted from our score for this problem).

We chose the MIP solver Gurobi for our experiments, which we ran on the NEOS server or under Linux Ubuntu 22.04.5 (64 bit) on an Intel Xeon E5520 of 2.27 GHz, with 4 processors of 4 cores each, with a 70 GiB RAM and an 8 MiB L3 cache (a ThinLinc computer of the IT department).

Tasks C, D, E, and F. The results are in Table 1.

Task G. The results are in Table 1. Our model does not time out.

Task H. The results are in Table 1. Our model times out, so our proposed algorithm for delivering a not necessarily optimal solution in reasonable running time is ... as follows ....

Task I. The size of the search space of the problem is $\binom{z!}{\cos c} \cdot \log_s v$, because ... justification ....

The numbers of candidate solutions this brute-force search algorithm has to examine per second in order to match the reported runtime performance of Gurobi on our model are given in the right-most column of Table 1, for each instance that Gurobi solved to proven optimality without timing out.

z	s	v	c	time	objective value	optimality gap	brute-force
10	2	2	3	67.89	0.008740338682	0.00%	10^{17}
10	3	2	3				
10	4	2	3				
20	2	2	3	123.45	0.023246261350	0.00%	10^{23}
20	3	2	3				
20	4	2	3				
20	5	2	3				
20	6	2	3				
40	5	2	3				
80	8	2	3				
80	16	1	3				
120	10	2	3				
250	12	3	4	314.56		2.34%	n/a

Table 1: Service station location: runtime (in seconds), objective value, and optimality gap (in percent; positive if an optimal solution was not found and proven before timing out) using Gurobi, with a timeout of 314.56 CPU seconds. The right-most column gives the number of candidate solutions a totally brute-force search algorithm has to examine per second in order to match the runtime performance of Gurobi, if the instance was solved to proven optimality, and 'n/a' for 'non-applicable' otherwise. (The sample performance of this skeleton table is made up, but the two given optimal objective values are correct!)

Problem 2: Stochastic Local Search (SLS)

Task A.

1. Representation. Describe how to represent the problem: what are the variables, their meanings, their constraints, and the objective function?

2. Initial Assignment. Describe an algorithm for generating (fast) a randomised initial assignment.

3. Move. Describe one or more moves that go from an assignment to a neighbouring assignment by changing the values of a few variables.

4. Constraints. Describe for each constraint how its satisfaction is either algorithmically checkable efficiently or guaranteed to be preserved by the previous two design choices.

5. Neighbourhood. Describe a neighbourhood based on the proposed moves. Derive a formula for computing the size of the neighbourhood in terms of the problem parameters. Discuss whether the neighbourhood makes the search space connected, in the sense that every feasible assignment (that is, every assignment satisfying all the constraints, whether optimal or not) is reachable from every initial assignment (you only need to sketch a proof if the search space is connected, and give a counterexample otherwise).

6. Cost Function. Describe a cost function, whose value is to be minimised during search.

7. Probing. Describe how a neighbouring assignment, as reachable by a move, can be probed efficiently: describe how the cost function can be evaluated efficiently and incrementally, and describe the data structures used to do so. Give, without proof, the time complexity of probing; ideally, it is (sub-)linear in the problem parameters.

8. Heuristic. Describe a heuristic for exploring (via probing) the neighbourhood and selecting a neighbouring assignment to commit to. State whether the neighbourhood is explored exhaustively and, if so, how you determine when it was exhausted. Explain how you ensure that the same neighbour is not probed twice during a given exploration.

9. Optimality. Describe how you use a bound on the objective value in order to terminate sometimes the search with proven optimality, as part of the heuristic.

10. Meta-Heuristic. Describe a meta-heuristic based on tabu search: explain how the tabu list is represented; choose (a formula for) its size; explain how fine-grained its content is; and describe how it can be looked up and maintained efficiently; note that the tabu list is not necessarily an actual list, but rather a concept; make sure that worsening moves are sometimes made.

11. Random Restarts. Describe how to detect or guess that a random restart should be made, as part of the meta-heuristic.

12. Optional Tweaks. Describe ideas that you have used in order to improve your algorithm.

In summary, our hyperparameters (not the problem parameters v, b, r) are α and β.

(We are aware that we may lose points if the algorithm described above is different from what is actually implemented in Task B and evaluated in Task C.)

Task B. We chose the high-performance programming language Java, for which a compiler or interpreter is available on the Linux computers of the IT department. All source code is uploaded with this report (but not listed inside it). The compilation and running instructions are ... as follows ....

We validated the correctness of our implementation by checking its outputs on how many instances via the provided polynomial-time solution checker.

Task C. All experiments were run under Linux Ubuntu 22.04.5 (64 bit) on an Intel Xeon E5520 of 2.27 GHz, with 4 processors of 4 cores each, with a 70 GiB RAM and an 8 MiB L3 cache (a ThinLinc computer of the IT department).

The median runtime (in seconds), median number of steps, and median achieved λ over 5 independent runs for each of the instances of the assignment instructions are given in Table 2, for two experimentally determined good configurations of values for our hyperparameters α and β. The timeout was 314.5 CPU seconds per run.

We observe that ... something happened ..., because ... justification ....

Task D. An exact algorithm could work as follows: discuss its features (for instance, does it perform brute-force search?). The size of the search space of this exact algorithm is $\binom{r!}{\cos b} \cdot \log v$, because ... justification ....

The number of candidate solutions this exact algorithm has to examine per second in order to match the runtime performance of the seemingly best configuration of values for our hyperparameters, according to Task C, of our stochastic local search algorithm is given in the right-most column of Table 2, for each instance solved to proven optimality. We think that ... this is amazing ..., because ... justification ....

Feedback to the Teachers

Please write a paragraph, which will *not* be graded, describing your experience with this assignment. You may also do so anonymously, by whichever channel you choose. Which tasks were

				$\langle\alpha,\beta\rangle = \langle 10, 5\rangle$			$\langle\alpha,\beta\rangle = \langle 20, 8\rangle$			
v	b	r	lb(λ)	time	steps	λ	time	steps	λ	exact
12	44	11	2	314.5	901	3	200.0	5678	2	10^{21}
15	21	7	2	314.5	1023	4	314.5	6789	3	n/a
19	19	9	4							
10	100	30	7							
11	150	50	14							
15	350	100	24							
10	360	120	32							

Table 2: Investment design: median runtime (in seconds), median number of steps, and median achieved λ, for two good configurations of values for our hyperparameters α and β, over 5 independent runs per instance, with a timeout of 314.5 CPU seconds per run. The right-most column gives the number of candidate solutions the outlined exact algorithm has to examine per second in order to match the runtime performance of the seemingly best configuration of values for our hyperparameters, namely $\langle\alpha,\beta\rangle = \langle 20, 8\rangle$, if the instance was solved to proven optimality, and 'n/a' for 'non-applicable' otherwise. (The sample performance of this skeleton table is made up!)

too difficult or too easy? Which tasks were interesting or boring? Recall that experiments are inevitable. This will help us improve the course for the next year.

More LaTeX and Technical Writing Advice

Unnumbered itemisation (only to be used when the order of the items does *not* matter):[1]

- Unnumbered displayed formula:
$$E = m \cdot c^2$$
- Numbered displayed formula, which is cross-referenced somewhere:
$$E = m \cdot c^2 \tag{1}$$
- Formula — the same as formula (1) — spanning more than one line:
$$\begin{gathered} E \\ = m \cdot c^2 \end{gathered}$$

Numbered itemisation (only to be used when the order of the items *does* matter):

1. First do this.
2. Then do that.
3. If we are not finished, then go back to Step 2, else stop.

Tables and elementary mathematics are typeset as exemplified in Table 3; see Short Math Guide for LaTeX for many more details.

Use `\mathit{...}` in mathematical mode for each multiple-letter identifier in order to avoid typesetting the identifier like the product of single-letter ones. For example, note the typographic difference between the identifier WL, obtained through `$\mathit{WL}$`, and the product WL, where there is a small space between the W and the L, obtained through `$WL$`.

Do *not* use programming-language-style lower-ASCII notation (such as ! for negation, && for conjunction, || for disjunction, and the equality sign = for assignment) in algorithms or formulas (but rather use $\neg$ or **not**, $\wedge$ or & or **and**, $\vee$ or **or**, and $\leftarrow$ or $:=$, respectively), as this testifies to a very strong confusion of concepts.

Figures can be imported with `\includegraphics` or drawn inside the LaTeX source code using the highly declarative notation of the `tikz` package: see Figure 1 for sample drawings. It is perfectly acceptable in this course to include scans or photos of drawings that were carefully done by hand.

If you are not sure whether you will stick to your current choice of notation or terminology, then introduce a new (possibly parametric) command. For example, upon

```
\newcommand{\Cardinality}[1]{\left\lvert#1\right\rvert}
```

the formula `$\Cardinality{S}$` typesets the cardinality of set S as $|S|$ with autosized vertical bars and proper spacing, but upon changing the definition of that parametric command to

```
\newcommand{\Cardinality}[1]{\# #1}
```

and recompiling, the formula `$\Cardinality{S}$` typesets the cardinality of set S as $\#S$.

You can thus achieve an arbitrary number of changes in the document with a *constant*-time change in its source code, rather than having to perform a *linear*-time find-and-replace operation within the source code, which is painstaking and error-prone. The imported file `macros.tex` has a lot of useful predefined commands about mathematics and algorithms.

[1]Use footnotes very sparingly, and note that footnote pointers are *never* preceded by a space and always glued immediately *behind* the punctuation, if there is any.

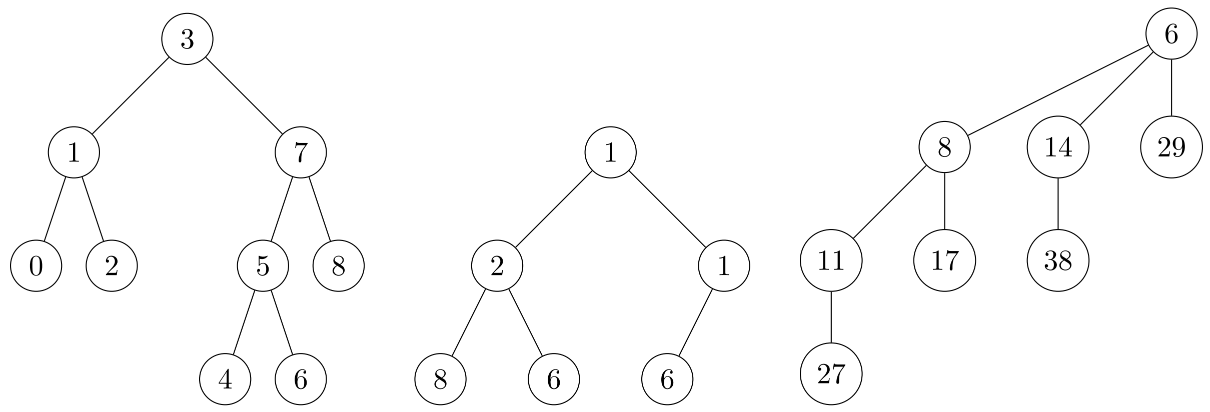

Figure 1: A binary search tree (on the left), a binary min-heap (in the middle), and a binomial tree of rank 3 (on the right).

Use commands on positioning (such as `\hspace`, `\vspace`, and `\noindent`) and appearance (such as `\small` for reducing the font size, and `\textit` for italics) very sparingly, and ideally only in (parametric) commands, as the very idea of mark-up languages such as LaTeX is to let the class designer (usually a trained professional typesetter) decide on where things appear and how they look. For example, `\emph` (for emphasis) compiles (outside italicised environments, such as `theorem`) into *italics* under the `article` class used for this document, but it may compile into **boldface** under some other class.

If you do not (need to) worry about *how* things look,
then you can fully focus on *what* you are trying to express!

Note that *no* absolute numbers are used in the LaTeX source code for any of the references inside this document. For ease of maintenance, `\label` is used for giving a label to something that is automatically numbered (such as an algorithm, equation, figure, footnote, item, line, part, section, subsection, or table), and `\ref` is used for referring to a label. An item in the bibliography file is referred to by `\cite` instead. Upon changing the text, it suffices to recompile, once or twice, and possibly to run BibTeX again, in order to update all references consistently.

Always write `Table~\ref{tab:maths}` instead of `Table \ref{tab:maths}`, by using the non-breaking space (which is typeset as the tilde ∼) instead of the normal space, because this avoids that a cross-reference is spread across a line break, as for example in "Table 3", which is considered poor typesetting.

The rules of English for how many spaces to use before and after various symbols are given in Table 4. Beware that they may be very different from the rules in your native language.

Feel free to report to the head teacher any other features that you would have liked to see discussed and exemplified in this document.

Topic	LaTeX code	Appearance
Greek letter	`$\Theta,\Omega,\epsilon$`	Θ, Ω, ϵ
multiplication	`$m \cdot n$`	$m \cdot n$
division	`$\frac{m}{n}, m \div n$`	$\frac{m}{n}, m \div n$
rounding down	`$\left\lfloor n \right\rfloor$`	$\left\lfloor n \right\rfloor$
rounding up	`$\left\lceil n \right\rceil$`	$\left\lceil n \right\rceil$
binary modulus	`$m \bmod n$`	$m \bmod n$
unary modulus	`$m \equiv n \mod \ell$`	$m \equiv n \mod \ell$
root	`$\sqrt{n},\sqrt[3]{n}$`	$\sqrt{n}, \sqrt[3]{n}$
exponentiation, superscript	`$n^{i}$`	n^{i}
subscript	`$n_{i}$`	n_{i}
overline	`$\overline{n}$`	$\overline{n}$
base 2 logarithm	`$\lg n$`	$\lg n$
base b logarithm	`$\log_b n$`	$\log_b n$
binomial	`$\binom{n}{k}$`	$\binom{n}{k}$
sum	`\[\sum_{i=1}^n i\]`	$\sum_{i=1}^n i$
numeric comparison	`$\leq,<,=,\neq,>,\geq$`	$\leq, <, =, \neq, >, \geq$
non-numeric comparison	`$\prec,\nprec,\preceq,\succeq$`	$\prec, \nprec, \preceq, \succeq$
extremum	`$\min,\max,+\infty,\bot,\top$`	$\min, \max, +\infty, \bot, \top$
function	`$f\colon A\to B,\circ,\mapsto$`	$f\colon A \to B, \circ, \mapsto$
sequence, tuple	`$\langle a,b,c \rangle$`	$\langle a, b, c \rangle$
set	`$\{a,b,c\},\emptyset,\mathbb{N}$`	$\{a, b, c\}, \emptyset, \mathbb{N}$
set membership	`$\in,\not\in$`	$\in, \not\in$
set comprehension	`$\{i \mid 1 \leq i \leq n\}$`	$\{i \mid 1 \leq i \leq n\}$
set operation	`$\cup,\cap,\setminus,\times$`	$\cup, \cap, \setminus, \times$
set comparison	`$\subset,\subseteq,\not\supset$`	$\subset, \subseteq, \not\supset$
logic quantifier	`$\forall,\exists,\nexists$`	$\forall, \exists, \nexists$
logic connective	`$\land,\lor,\neg,\Rightarrow$`	$\land, \lor, \neg, \Rightarrow$
logic	`$\models,\equiv,\vdash$`	$\models, \equiv, \vdash$
miscellaneous	`$\&,\#,\approx,\sim,\ell$`	$\&, \#, \approx, \sim, \ell$
dots	`$\ldots,\cdots,\vdots,\ddots$`	$\ldots, \cdots, \vdots, \ddots$
dots (context-sensitive)	`$1,\dots,n; 1+\dots+n$`	$1, \dots, n; 1 + \dots + n$
parentheses (autosizing)	`$\left(m^{n^k}\right),(m^{n^k})$`	$\left(m^{n^k}\right), (m^{n^k})$
identifier of $>$ 1 character	`$\mathit{identifier}$`	$\mathit{identifier}$
hyphen, n-dash, m-dash, minus	`-, --, ---, $-$`	-, –, —, $-$

Table 3: The typesetting of elementary mathematics. Note very carefully when italics are used by LaTeX and when not, as well as all the horizontal and vertical spacing performed by LaTeX.

		number of spaces after	
		0	1
number of spaces before	0	/ -	, : ; . ! ?)] } ’ ” %
	1	([{ ‘ “	– (n-dash) — (m-dash)

Table 4: Spacing rules of English