Modelling for Combinatorial Optimisation (1DL451) Uppsala University – Autumn 2025 Report by Team 22: Minimal Magic Square

Clara CLÄVER and Whiz KIDD

4th June 2025

(The usage of LaTeX is *optional*, but highly recommended, for reasons that will become clear to those who have never used it before, for example because of our experiment-running script. Any learning time of LaTeX is *outside* the time budget of this course, but will hugely pay off, if not in this course then in your next courses and when writing a thesis or other report.)

A Minimal Magic Square

(***Only for the project report***: Describe the problem in your own words, citing the source of the problem and of every included third-party picture.)

Minimal Magic Square is about finding a magic square of a given order, with minimal sum of the corner elements. A *magic square* of order n is an $n \times n$ array containing all the integers 1 to n^2 exactly once, so that the sums of the rows, columns, and main diagonals are all equal.

B Approach

(***Only for the project report***: Describe your approach to your project problem. If your approach is just a single model (like for the assignments), then just say so and only follow the model instructions below, else describe your software pipeline — with pre-processing, solving (possibly on a sequence of models), and post-processing — and follow the model instructions below for ***each*** model. Either way, upload ***all*** code and data to Studium (***except*** for the initial report), but ***only*** import your model(s) into the report itself. The report must be about exactly ***one*** approach (and this word is ***not*** a synonym for 'viewpoint') and must ***not*** describe all false starts. If you want or need to use the MiniZinc-Python interface for the project (do ***not*** use it for the assignments), then our instructions help you get started, with the example of a toy model. Also note that there is JSON support (see Section 4.1.13).)

Our approach is to write a single model, without pre-processing and post-processing outside the MiniZinc toolchain. The model is given in Listing 1: it has the prescribed comments (see below) as per the scope (Topics 1 to 8) of the project, it has the name `minMagicSquare.mzn` and the imposed structure of the provided skeleton model `project-skeleton.mzn`, and it is uploaded to Studium.

(Each MiniZinc [1] model ***must*** include comments that give each of the following explanations, provided the scope of the assignment goes at least until the indicated topic, the scope of the project being Topics 1 to 8:

- the meaning of each parameter (that is in the provided skeleton model);

- the meaning of each derived parameter (that is not in the provided skeleton model);
- the meaning of each decision variable;
- a paraphrase of each problem constraint enforced by the choice of decision variables;
- the meaning and (non-)mutuality of each redundant decision variable, or a justification why there are none (Topic 4);
- a 'one-way' or 'two-way' indication for each channelling constraint, or a justification why there are none (Topic 4);
- a paraphrase of each problem constraint not enforced by the choice of decision variables;
- a paraphrase of each implied constraint (Topic 4), or an indication that there are none, with a discussion in the **Efficiency** explanation;
- a paraphrase of each symmetry-breaking constraint and an indication whether it breaks all or some of its targeted symmetries (Topic 5), or an indication that there are none, with a discussion in the **Symmetries** and **Efficiency** explanations;
- a paraphrase of the objective and (if there is one) the objective function;
- the impact of each reasoning annotation (Topic 8), with further discussion in the **Efficiency** explanation;
- a paraphrase of each search annotation (Topic 8), with discussion in the **Efficiency** explanation.

The quality of comments in a model is considered while grading.)

Listing 1: A MiniZinc model for Minimal Magic Square

```
%% Parameters:
int: n; % width (and height) of the magic square

%% Derived parameters:
set of int: N = 1..n; % index range for the rows and columns
% The magic sum, for each row, column, and major diagonal:
int: magicSum = sum(1..n*n) div n;

%% Decision variables:
% Magic[r,c] = value at row r and column c of the magic square:
array[N,N] of var 1..n*n: Magic;
% No problem constraints are enforced by this choice of variables.
% (See the slides for examples)

%% Redundant decision variables:
% None.
% (See the slides for examples, marked as mutual or non-mutual.)

%% Channelling constraints:
% None, because there are no redundant decision variables.
% (See the slides for examples, marked as 1-way or 2-way.)

%% Problem constraints:
% All values in the magic square are distinct:
```

```
constraint all_different(Magic);
% Each row sums up to the magic sum:
constraint forall(r in N)(sum(Magic[r,..]) = magicSum);
% Each column sums up to the magic sum:
constraint forall(c in N)(sum(Magic[..,c]) = magicSum);
% The major down-diagonal sums up to the magic sum:
constraint sum([Magic[    i,i] | i in N])  = magicSum;
% The major up-diagonal sums up to the magic sum:
constraint sum([Magic[n+1-i,i] | i in N])  = magicSum;

%% Implied constraints:
% None.
% (See the slides for examples.)

%% Symmetry-breaking constraints:
% Break all 4 rotation symmetries and 4 reflection symmetries:
constraint symmetry_breaking_constraint(Magic[1,1] < Magic[1,n] /\
  Magic[1,n] < Magic[n,1] /\ Magic[1,1] < Magic[n,n]);

solve
  %% Search strategy:
  % Search (1) on the four corners -- which are subject to the
  % objective function (below) and more constraints (diagonals and
  % symmetry) than the other cells -- starting in the lower halves
  % of bisections of their domains, due to the minimisation and
  % the arithmetic in the constraints; and (2) similarly for the
  % remaining cells, starting with those on the major diagonals:
  :: seq_search([
       int_search([Magic[i,j] | i,j in {1,n}],
         input_order,indomain_split),
       int_search(Magic,occurrence,indomain_split)
     ])
  %% Objective and objective function:
  % Minimise the sum of the four corners of the magic square:
  minimize sum([Magic[i,j] | i,j in {1,n}]);

%% Pretty-print solutions:
output ["Magic sum: \(magicSum) \nCorner sum: \(sum([Magic[i,j] |
  i,j in {1,n}])) \n"] ++ [show2d(Magic)];
```

Symmetries. (Identify the *problem symmetries*, which exist for every viewpoint. Identify the *model symmetries*, which are introduced by your viewpoint. For each identified symmetry, state whether it is a *variable symmetry* (an example of which is an *index symmetry*, say a *row symmetry* or a *column symmetry*) or a *value symmetry*, and whether it is a *full symmetry* or a *partial symmetry*.)

The **problem symmetries** are the four rotation symmetries and the four reflection symmetries of a square, which are variable symmetries under our viewpoint and for which the qualifiers ‘full’ and ‘partial’ do not make sense. There is no value symmetry in our model. No **model symmetries** are introduced by our viewpoint.

Efficiency. (State the impact, or justify the absence, of implied constraints and symmetry-breaking constraints; do ***not*** test them on CBLS backends, such as Yuck, which ignore them. State the impact, or justify the absence, of reasoning annotations and search annotations: one ***only*** needs to test them on backends that use CP algorithms – such as Gecode, Chuffed, and CP-SAT – and you can restrict your tests to the CP backend Gecode.)

We identified no **implied constraints** that accelerate our model; an initial idea was to declare `magicSum` as a decision variable, with `n * magicSum = sum(1..n*n)` as an implied constraint, but this accelerates all solvers since it uniquely determines `magicSum` from parameter `n`, hence we realised that `magicSum` actually is a derived parameter, as declared in line 11 of our model.

The impact on Gecode of the **symmetry-breaking constraints** in line 45 is huge: already for `n=5`, an optimal solution is found and proven in 47 seconds instead of timing out when allocating 3 minutes (under the search annotation mentioned below).

We tried under Gecode the **reasoning annotation** `domain_propagation` on the constraint `all_different(Magic)` of line 29, but it seems the weaker `bounds_propagation` is performed in this case, and is faster.

The impact on Gecode of the **search annotation** in lines 55–58 is huge: already for `n=5`, an optimal solution is found and proven in 47 seconds instead of timing out when allocating 3 minutes (under the symmetry-breaking constraints mentioned above).

Checklist. (For each model feature covered by advice in the checklists of Topics 2 & 3 (if Topic 3 is in the announced scope of the current assignment): how do you argue that it does not matter? For example, for a `where` clause involving decision variables, does a profiled compilation reveal numbers of generated variables and constraints that you argue to be acceptable; or is the solving time comparable to the one of an uncovered reformulation that you give?)

Our model has no features covered by advice in the checklists of Topics 2 and 3.

Correctness. (***Only for the project report***: How do you argue that your approach is correct? For example, do you use a checker based on another model, such as a model by another team or a linearisation of a MIP model found somewhere? Do you use the same new instances as another team? If it is a satisfaction problem: do you compare the numbers of solutions with numbers found somewhere or reported by another team? If it is an optimisation problem: do you compare objective values that are proven optimal before timing out (respectively objective values known when timing out) with optimal objective values (respectively best-known objective values) found somewhere or reported by another team?)

All the objective values in Table 1 that were proven minimal before timing out agree with those reported at `https://en.wikileaks.org/wiki/Minimal_magic_square`.

C Evaluation

(Hint: Under Linux, do `lscpu` to find CPU information. Under macOS, you find CPU information via "About This Mac" in the Apple menu.)

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).

(You ***must*** use the script of our cheatsheet: it conducts the experiments and generates a result table (see the LaTeX source code of Table 1) that is automatically imported (rather than manually copied) into your report, so each time you change the model, it suffices to re-run that script and re-compile your report, without any tedious number copying!)

Table 1 gives our results for various values of parameter `n` in our model. The time-out was 3,000,000 milliseconds.

Backend	Chuffed		CP-SAT		Gecode		Gurobi		PicatSAT		Yuck	
n	obj	time	obj	time	obj	time	obj	time	obj	time	obj	time
3	**20**	**1**	**20**	585	**20**	606	**20**	684	**20**	693	**20**	t/o
4	**34**	**180**	**34**	696	**34**	646	**34**	1123	**34**	764	**34**	t/o
5	**26**	381910	**26**	**2594**	**26**	47128	**26**	65629	**26**	t/o	**26**	t/o
6	-	t/o	27	t/o	-	t/o	**26**	**296077**	28	t/o	**26**	t/o

Table 1: Results for our model of Minimal Magic Square, which is a minimisation problem. In each `time` column: if the reported time is less than the time-out (3,000,000 milliseconds here), then the objective value in the corresponding `obj` column was *proven* optimal; else the timing out is indicated by `t/o` and the objective value is either the best one found but *not* proven optimal before timing out, or '`-`' indicating that no feasible solution was found before timing out. Boldface indicates the best performance (time or objective value) on each row.

Which backends win overall, and how do you draw that conclusion? We observe that Gurobi wins overall, as it is the only backend to prove optimality for the largest chosen instance before timing out. However, note that Yuck also finds all the optima.

How do the backends scale, and how do you draw that conclusion? On the smaller instances, with $n \leq 4$, Chuffed clearly wins, all others finding the optima in similar times to each other, except that Yuck necessarily times out (see below for the reason), even though it actually also finds the optima. On the medium instance, with $n = 5$, CP-SAT takes over and clearly wins, with now also PicatSAT unable to prove the optimum. On the largest instance, with $n = 6$, Gurobi takes over and clearly wins, with only Yuck still finding the optimum, and with Chuffed and Gecode now unable to establish even only feasibility.

Does the difficulty of instances monotonically increase with their size, and how do you draw that conclusion? For all backends, difficulty seems to increase with `n`. Interestingly, observe in the Gurobi `obj` column that the provably optimal objective value does not monotonically increase with `n`.

How suitable is local search compared to systematic search, and how do you draw that conclusion? Yuck always times out, because local search can by construction not prove minima on problems, such as here, where the trivial lower bound (namely $10 = 1 + 2 + 3 + 4$ for all instances here) on the objective value is not feasible. However, Yuck finds all the optima proven by Gurobi! We conjecture that Yuck scales better than Gurobi on even larger instances.

Are there any contradictions between the results? No results are contradictory. All proven optima are the same.

Are there any occurrences of 'ERR' within the results generated by the experiment script? (If so, then first try and troubleshoot on your own by running the incriminated backend manually (within the IDE or at the command line by using the `--solver` flag of the `minizinc` command) and interpreting the error message. If you cannot resolve the error, then you ***must*** state here for each occurrence of 'ERR' when and how you received a teacher's ***prior*** approval to include it, and you ought to make an error report in the final section.)

No occurrences of 'ERR' were generated by the experiment script.

Feedback to the Teachers

(Please write a paragraph, which will ***not*** be graded, describing your experience with this report: which aspects were too difficult or too easy, and which aspects were interesting or boring? This will help us improve the course for the next year.)

Error Report

(Your model ***must*** compile and run error-free under backends of ***all*** the considered solving technologies, unless you have a teacher's ***prior*** approval to upload an error report here.)

largecumulative.mzn We secured the head teacher's oral permission on 7 September 2024 to upload this error report: for the larger instances, fzn-oscar-cbls [which is no longer used in this course, and which was fixed after this error report] crashes with the following exception, which seems to be caused by the JVM not being allocated enough heap space:

```
>./fzn-oscar-cbls -s -t 600 /tmp/tmp.fzn
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
  at scala.collection.mutable.FlatHashTable$class.growTable(FlatHashTable.scala:217)
  at scala.collection.mutable.FlatHashTable$class.addEntry(FlatHashTable.scala:159)
  at scala.collection.mutable.HashSet.addEntry(HashSet.scala:40)
  at scala.collection.mutable.FlatHashTable$class.addElem(FlatHashTable.scala:139)
  at scala.collection.mutable.HashSet.addElem(HashSet.scala:40)
  at scala.collection.mutable.HashSet.$plus$eq(HashSet.scala:59)
  at oscar.cp.scheduling.constraints.EnergeticReasoning$$anonfun$computeIntervals$1$$anonfun$apply$mcVI$sp$2.apply(EnergeticReasoning.scala:127)
  at oscar.cp.scheduling.constraints.EnergeticReasoning$$anonfun$computeIntervals$1$$anonfun$apply$mcVI$sp$2.apply(EnergeticReasoning.scala:126)
  at scala.collection.TraversableLike$WithFilter$$anonfun$foreach$1.apply(TraversableLike.scala:778)
  at scala.collection.mutable.HashSet.foreach(HashSet.scala:78)
  at scala.collection.TraversableLike$WithFilter.foreach(TraversableLike.scala:777)
  at oscar.cp.scheduling.constraints.EnergeticReasoning$$anonfun$computeIntervals$1.apply$mcVI$sp(EnergeticReasoning.scala:126)
  at oscar.cp.scheduling.constraints.EnergeticReasoning$$anonfun$computeIntervals$1.apply(EnergeticReasoning.scala:126)
  at oscar.cp.scheduling.constraints.EnergeticReasoning$$anonfun$computeIntervals$1.apply(EnergeticReasoning.scala:126)
  at scala.collection.mutable.HashSet.foreach(HashSet.scala:78)
  at oscar.cp.scheduling.constraints.EnergeticReasoning.computeIntervals(EnergeticReasoning.scala:126)
```

References

[1] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In Christian Bessière, editor, *CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007. The MiniZinc toolchain is available at https://www.minizinc.org.

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 2; 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.

Algorithms can be typeset as pseudo-code as exemplified in Algorithm 1: study its LaTeX source code.

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

[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.

```
function $f(n)$
if $n < 0$ then                                  // optional comment
    $n := -2 \cdot n$                            // optional comment
else                                             // $n \geq 0$
    $n := 3 \cdot n$
while $n > 0$ do                                 // optional comment
    $n := n - 1$
return $n$
```

Algorithm 1: Silly algorithm

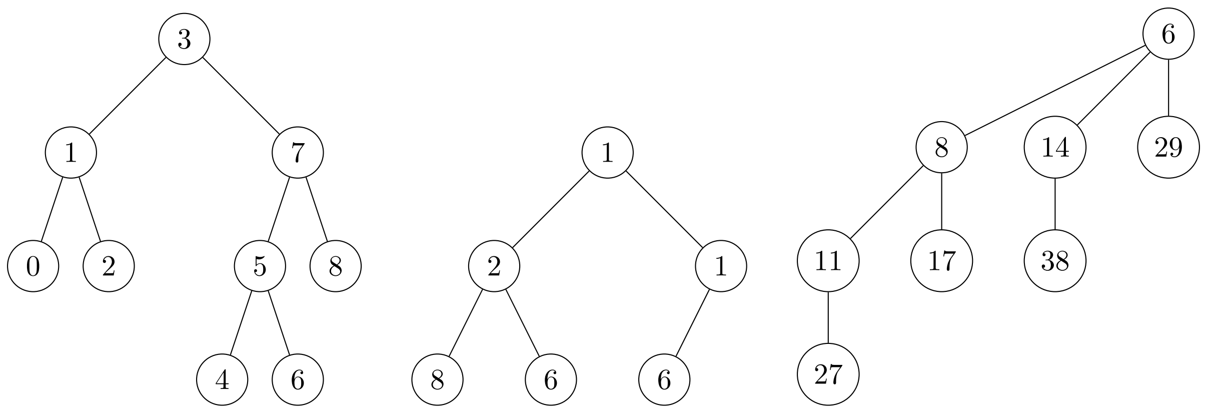

Figure 1: A binary search tree (on the left), a binary min-heap (in the middle), and a binomial tree of rank 3 (on the right).

```
\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 $\#S$. Similarly, upon

```
\newcommand{\MiniZinc}{\textit{Mini\-Zinc}}
```

the text `\MiniZinc\` typesets into *MiniZinc*, hyphenation being only possible in the middle, but upon changing the definition of that non-parametric command to

```
\newcommand{\MiniZinc}{\textsc{Mini\-Zinc}}
```

and recompiling, the text `\MiniZinc\` typesets into MINIZINC. You can thus obtain 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, CP, Gecode, modelling, MiniZinc, and algorithms.

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 2", which is considered poor typesetting.

The rules of English for how many spaces to use before and after various symbols are given in Table 3. 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 template 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 2: 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 3: Spacing rules of English