"The Four Strongest" Coursework for MATH3082

Dan Roberts, *Student, University of Southampton, dr1n22*

I. Introduction

THE 'Four Strongest' puzzle at the National Museum of Mongolia is a traditional Mongolian game that tasks fitting small statues of four legendary animals; dragon D, tiger T, lion L, and garuda G (a Buddhist mythological bird) , into an elongated hexagonal wooden container. The challenge comes in finding a correct orientation and order to place the statues in, so that they fit inside of the wooden container. A trivial approach to solving this problem is to test all possible configurations of the animals and select any (or all) that successfully fit in the container without overlap.

While the number of unique orders is quite small ($4! = 24$), the number of possible orientations is infinite, since rotating a given animal an infinitely small amount yields a new distinct orientation. If we consider only 360 possible orientations, we still have to check $\frac{4! \times 360^4}{2} \approx 2 \times 10^{11}$ different possible combinations (accounting for mirror symmetry). Clearly, a brute-force method is not feasible, and an alternative approach is required.

A. Previous work

Ninjbat (2020) reduce the number of orientations to 2; 'up' (head of the animal facing forward) or 'down' (rotated 180°). This (strong) assumption reduces the number of possible configurations to just 192, which Ninjbat exhaustively explores to yield a valid configuration of:

$$D\uparrow, L\downarrow, T\uparrow, G\uparrow$$

Singh (2024) formulates the Four Strongest puzzle as a mathematical optimisation model using linear programming. Similarly to Ninjbat, Singh also considers only 'up' and 'down' orientations and uses approximate measurements for the dimensions of the various animals. Using binary decision variables to dictate the orientation and order of the various statues, Singh achieves a valid configuration without the need for exhaustive search:

$$T\uparrow, D\uparrow, L\uparrow, G\downarrow$$

B. Structure

In this report, we critically review Singh's formulation of an optimisation model, addressing the underlying assumptions of the model and exploring the impact of relaxing or removing them. We propose an alternative linear programming model to solve the Four Strongest puzzle and compare the results to those of Ninjbat and Singh. Finally, we conclude with a critical evaluation of the proposed model against Singh's.

II. Review of Singh's Model

Singh formulates the Four Strongest puzzle as a binary linear program with the objective of minimizing the approximate total length of the animals.

In this model, Singh defines three sets of decision variables:

$$I = \{D, T, L, G\}, \quad J = \{\text{'up'}, \text{'down'}\}, \quad K = \{1, 2, 3, 4\}$$

where $x_{i,j,k} = 1$ if animal i is placed at position k with orientation j, otherwise $x_{i,j,k} = 0$ ($i \in I, j \in J, k \in K$). Singh tracks the relative position of pieces using $y_{i,j,k,i',j',k+1} = 1$ if animal i is placed at position k with orientation j and is immediately followed by (i.e., placed before) animal i' at position $k+1$ with orientation j', otherwise 0.

The objective function measures the total distance between adjacent pairs of pieces (first sum in Equation 1). To correct for the implicit double-counting of interior pieces (specifically, those at positions 2 and 3) in the adjacency measurements, the width of the interior pieces are subtracted (second sum), yielding an approximate total length for a given configuration.

$$\min \sum_{\substack{i,i' \in I \\ j,j' \in J \\ k \in K \\ i \neq i',\ k<|K|}} d_{i,j,i',j'} \cdot y_{i,j,k,i',j',k+1} - \sum_{\substack{i \in I \\ j \in J \\ 1<k<|K|}} b_{i,j} \cdot x_{i,j,k} \tag{1}$$

The following constraints are applied to the objective function to ensure a valid configuration:

1) Position constraint: exactly one piece (with some orientation j) is placed at each position. Without this constraint, the trivial solution—placing no animals at all—would minimize the total length.

$$\sum_{i \in I} \sum_{j \in J} x_{i,j,k} = 1 \quad \forall k \in K$$

2) Animal constraint: each animal is placed exactly once. Without this constraint, the optimisation could select the shortest animal multiple times—potentially placing it in all four positions—to minimize the total arrangement length.

$$\sum_{j \in K} \sum_{k \in K} x_{i,j,k} = 1 \quad \forall i \in I$$

3) Adjacency constraint: The McCormick envelope formulation (**?**) is employed to linearize the adjacency variables, which originally involve bilinear terms. Without this linearization, the model would become

non-convex and require more complex solution methods.

$$\left.\begin{aligned} &y_{i,j,k,i',j',k+1} \leq x_{i,j,k} \\ &y_{i,j,k,i',j',k+1} \leq x_{i',j',k+1} \\ &y_{i,j,k,i',j',k+1} \geq x_{i,j,k} + x_{i',j',k+1} - 1 \end{aligned}\right\} \begin{aligned} &\forall i, i' \in I; \\ &\forall j, j' \in J; \\ &\forall k \in K; \\ &i \neq i', \quad k < |K| \end{aligned}$$

4) Binary constraints: The placement variables x and adjacency variables y are restricted to binary values. Without enforcing binary constraints, the model could assign fractional placements to pieces and orders, which would not correspond to any physical configuration.

$$x_{i,j,k} \in \{0,1\}, \quad y_{i,j,k,i',j',k+1} \in \{0,1\}$$

A. Assumptions

Singh, like Ninjbat, introduces several assumptions about the structure and data of the problem that simplify it to make it solvable, but also move it away from the true physical puzzle.

1) Only two orientations: The use of only 'up' and 'down' orientations greatly simplifies the puzzle and reduces the number of possible combinations from infinite to manageable, but is far from the reality of orienting the animals in almost any way. If we were to allow for a larger number of orientations (e.g. every 5° or 10°), it would massively increase the number of configurations, making the model vastly larger and more complicated, and potentially unsolvable using traditional linear programming techniques.

2) Container shape approximation: Approximating the container as a rectangle simplifies the model by allowing lengths to be measured along a single dimension. However, this assumption diverges from the true physical puzzle, which involves fitting pieces into a hexagonal box. Accurately modelling the original container would require formulating a two-dimensional geometric packing problem, a type of optimisation problem known for its computational complexity (Delorme and Iori (2016)).

3) Approximate measurements: Singh did not have access to the physical animal statues. They use approximate measurements obtained from scaled photographs of the bases of the various animals. If the true measurements were used, the model may arrive at a different optimal configuration.

4) Linear 1-dimensional adjacency placement: The animals are assumed to be placed end-to-end in a straight line, as this simplifies the adjacency calculations. If the animals are allowed to be placed spatially, it would add another dimension to the animal position (x and y positions) and again would become a packing problem.

5) Inclusion-Exclusion principle: Measurements are only made for pairs of animals; measurements for 3- or 4-animal configurations are approximated based on adjacent pairs. This avoids the need for exponentially many individual measurements needing to be made, at the cost of reduced accuracy of the model.

B. Strengths

- Use of standard linear programming solvers: The use of GAMS allows the formulation to be reproducible, easily implemented and widely accessible.
- High quality example: The model is a very good example of a linear programming problem, and how it is formulated.
- Efficient formulation: Allows a valid solution to be found without the need for exhaustive search. Particularly powerful if the number of animals or orientations grows.

C. Weakness

- Strict no over-lapping: By strictly enforcing that animals cannot overlap, the model is unable to capture the 'tight fit' that is possible in reality but exploiting the infinite number of orientations and the irregular shapes of the animals.
- Inclusion-exclusion principle: Although the use of this measurement approximation is clever, it can be inaccurate for tightly packed configurations. Furthermore, additional animals require new measurements to be made for every new pair.

III. The Model

We adopt a geometric approach that directly models the physical layout of the animals in space, staying closer to the true nature of the puzzle. Each animal is assigned a starting position as a continuous value along a 1D axis, allowing the problem to be formulated as a one-dimensional spatial packing problem (considerably easier to solve than its two-dimensional counterpart). This avoids the need for the inclusion-exclusion principle used by Singh, and reduces the reliance on approximated pairwise measurements. We maintain the following assumptions from Singh's work: 1) Only two orientations, 2) Container shape approximation, and 3) Approximate measurements.

Since orientation and shape variations can significantly affect how tightly the animals fit, we allow for small overlaps of animals (up to a defined tolerance δ). This models realistic 'tight fits' that are possible by experimenting with the infinite number of orientations possible in the real puzzle, where slight contact and misalignment is still an acceptable solution. We model the physical container by minimising the rightmost endpoint of the configuration, as this is more intuitive than the inclusion-exclusion principle used by Singh.

A. Alternate Formulation

We use the same definitions of I, J, K, $b_{i,j}$ as defined by Singh. Let $s_i \in \mathbb{R}_{\geq 0}$ denote the starting position of toy i (in cm). The first toy ($x_{i,1} = 1$) will have $s_i = 0.0$cm. Let δ denote a fixed maximum allowable overlap between toys (e.g., 0.25cm). If toy i and i' are placed adjacently, toy i's end can extend into i''s space by up to δcm. We then have two binary variables; $x_{i,k} = 1$ if toy i is assigned position k, else 0; and $y_{i,j} = 1$ if toy i has orientation j, else 0. A summary

TABLE I
SUMMARY OF SYMBOLS USED IN MODEL

Symbol	Definition
I	Set of animals: $\{D, T, L, G\}$
J	Set of orientations: $\{\text{up}, \text{down}\}$
K	Set of positions: $\{1, 2, 3, 4\}$
$x_{i,k}$	1 if animal i is placed in position k, else 0
$y_{i,j}$	1 if animal i is assigned orientation j, else 0
s_i	Start position of animal i (in cm)
$b_{i,j}$	Length of animal i in orientation j (in cm)
δ	Fixed maximum overlap tolerance (e.g., 0.25 cm)
M	Large constant used for Big-M constraint logic
L	Total length of the arrangement (rightmost endpoint)

of symbols used can be seen in Table I. Our objective is to minimise the total length used by the configuration:

$$\min \quad L \tag{2}$$

where L is the rightmost edge of the last animal. We have the following constraints to ensure validity:

1) Position constraint I: Each animal must occupy exactly one position.

$$\sum_{k \in K} x_{i,k} = 1 \quad \forall i \in I$$

2) Position constraint II: Each position must be filled by exactly one toy.

$$\sum_{i \in I} x_{i,k} = 1 \quad \forall k \in K$$

3) Orientation constraint: Each toy must have exactly one orientation.

$$\sum_{j \in J} y_{i,j} = 1 \quad \forall i \in I$$

4) Overlap constraint: The amount of overlap between toys i and i' must be at most δcm, if i is placed directly before i'.

$$s_i + \sum_{j \in J} b_{i,j} \cdot y_{i,j} \leq s_{i'} + \delta + M \cdot (2 - x_{i,k} - x_{i',k+1})$$
$$\forall k \in \{1, \ldots, |K| - 1\}, \quad \forall i, i' \in I, i \neq i'$$

This constraint is enforced only if toy i is in position k and toy i' is placed in position $k+1$; otherwise, the constraint is relaxed using the Big-M method.

5) Length constraint: The objective variable must be at least as far as the right-most end of each toy.

$$s_i + \sum_{j \in J} b_{i,j} \cdot y_{i,j} \leq L \quad \forall i \in I$$

6) Non-negativity and domains:

$$s_i \geq 0, L \geq 0, \quad \forall i, i' \in I, i \neq i'$$
$$x_{i,j} \in \{0, 1\}, \quad y_{i,j} \in \{0, 1\}$$

We need to enforce Constraint 4 only if toy i is placed in position k and toy i' is placed in position $k+1$, otherwise we do not want this constraint to be applied. Standard linear constraints cannot encode this type of logic. We use the Big-M method to convert this logical expression to a mathematical expression.

If toy i is in position k ($x_{i,k} = 1$) and toy i' is in position $k+1$ ($x_{i',k+1} = 1$), the expression $M \cdot (2 - x_{i,k} - x_{i',k+1})$ becomes 0, simplifying Constraint 4 to:

$$s_i + \sum_{j \in J} b_{i,j} \cdot y_{i,j} \leq s_{i'} + \delta$$

If either toy i or i' is not placed in positions k or $k+1$ respectively, the expression becomes:

$$s_i + \sum_{j \in J} b_{i,j} \cdot y_{i,j} \leq \textit{large number}$$

which is always true, making this constraint effectively disabled. The value of M should be large enough to deactivate the constraint when required, but not too large as to cause numerical instability for the solver.

IV. RESULTS & COMPARISON

We use the Python library `PuLP` as the solver for our model Mitchell (2011). `PuLP` is an linear and mixed-integer programmer, and was chosen for its user-friendly syntax. Full code can be seen in Appendix A.

By solving the model, we arrive at a feasible solution of:

$$D \downarrow, G \downarrow, T \downarrow, L \downarrow$$

This gives a total final length of 31.06cm. We can calculate the length according to the inclusion-exclusion principle as described by Singh, and arrive at a length of 31.58cm; a discrepancy of approximately 1.65%.

Although our calculated configuration is 'worse' than Singh's (at 29.08cm), it is almost identical to the solution proposed by Ninjbat, which had a length of 31.54cm. Our solution matches more closely with the calculated length of the real-world solution provided by Ninjbat, though the order does differ.

The difference in solution could be for a variety of reasons, including the introduction of a tolerance δ, the choice of objective function, or the approximation of measurements used by both our model and Singh's. Ultimately, the best way to verify if a solution is valid is to physically use the containers and animals themselves.

A. Model Comparison

Our model provides an alternate formulation of the Four Strongest puzzle as an optimisation problem. We use many of the same assumptions and definitions as Singh, and our model finds a solution that is close to the real-world solution found by Ninjbat.

1) Assumptions: Our model directly encodes the physical layout of the animals within the container by using their lengths and a starting position, and allows small overlaps via a fixed tolerance δ to model the real-world 'tight fit' that is limited by the use of only two orientations. Singh's model strictly does not allow any overlap between adjacent animals through the use of approximated pairwise measurements. To this end, Singh's model would require additional pairwise measurements to be taken for every new pair if more toys were to be added to the puzzle, whereas our model would require only the measurement of each new toy. This allows the number of measurements required by our solution to grow with $\mathbf{O}(n)$, rather than $\mathbf{O}(n^2)$ for Singh's, where n is the number of animals.

2) Performance: Both our model and Singh's find an optimal solution in a matter of seconds when run on a Macbook Air M1.

3) Exercise 4: This exercise proposed by Singh asks to describe another method to estimate the width b and pairwise distances d of each animal. We propose a physics-engine based approach, using an engine such as Unity.

Each animal can be modelled as a 2D polygonal object using the silhouette provided in Ninjbat Figure 6. Unity can detect collisions between objects automatically and can adjust if needed. The objects can be converted to physical units using a calibrated scale, such as the length of the container as shown in Ninjbat Figure 5. Unity can provide measurements for various aspects of each animal, such as its area and perimeter, and could be extended to calculate b and d for additional orientations and more animals.

V. Conclusion

Our model provides a more intuitive formulation of an optimisation approach to the Four Strongest puzzle. By avoiding the inclusion-exclusion principle used by Singh, our model generalises better for additional animals, and better reflects the 'tight fit' that is possible in reality by using all possible rotations of the animals within the container. However, our model is still quite far removed from reality as it maintains the assumptions made by Singh that the container is a one-dimensional line and is still limited to two orientations. There is ample space for future work on this puzzle, including formulation as a two-dimensional packing problem, additional animals and orientations, and physics-engine-driven measurements.

References

Delorme, M. and Iori, M. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20.

Mitchell, S. D. (2011). Pulp: A linear programming toolkit for python. https://coin-or.github.io/pulp/. Version 2.7.0, accessed May 2025.

Ninjbat, U. (2020). The four strongest at the national museum of mongolia. *The Mathematical Intelligencer*, 42(2):9–14.

Singh, B. (2024). A mathematical programming approach for mongolia's "the four strongest" puzzle. *INFORMS Transactions on Education*. Articles in Advance.

Appendix A
Python Code

```python
import pandas as pd
import pulp
from itertools import permutations

# === Load data ===
df = pd.read_csv("breadths.csv")

# === Map animal names to short codes ===
name_map = {
    'dragon': 'D',
    'tiger': 'T',
    'lion': 'L',
    'garuda': 'G',
}

arrow = {
    "up": "\u2191",
    "down": "\u2193"
}

# === Build b[i, j] dictionary with full orientation info ===
b = {}
for _, row in df.iterrows():
    animal = name_map[row['animal'].strip().lower()]
    orientation = row['orientation'].strip().lower()  # 'up' or 'down'
    b[(animal, orientation)] = float(row['value'])

# === Static data ===
TOYS = ['D', 'T', 'L', 'G']
ORIENTATIONS = ['up', 'down']
POSITIONS = list(range(len(TOYS)))
M = 100
TOLERANCE = 0.25  # in cm

# === Model ===
model = pulp.LpProblem("FourStrongest", pulp.LpMinimize) # Min

# x[i,k]: is toy i in position k?
x = pulp.LpVariable.dicts("x", [(i, k) for i in TOYS for k in POSITIONS], cat='Binary')

# j_i[i,j]: is toy i in orientation j?
j_i = pulp.LpVariable.dicts("j", [(i, j) for i in TOYS for j in ORIENTATIONS], cat='Binary')

# s[i]: starting position of toy i
s = pulp.LpVariable.dicts("s", TOYS, lowBound=0)

rightmost = pulp.LpVariable("rightmost", lowBound=0)

# === Constraints ===

# 1. Each toy must appear in one position
for i in TOYS:
    model += pulp.lpSum(x[i, k] for k in POSITIONS) == 1

# 2. Each position must be occupied by exactly one toy
for k in POSITIONS:
    model += pulp.lpSum(x[i, k] for i in TOYS) == 1

# 3. Each toy has exactly one orientation
for i in TOYS:
    model += pulp.lpSum(j_i[i, j] for j in ORIENTATIONS) == 1
```

```
# 4. Overlap constraints based on orientation-dependent lengths (with tolerance)
for i, i2 in permutations(TOYS, 2):
    for k in POSITIONS[:-1]:  # positions 0, 1, 2
        # Only apply when i is in position k and i2 in k+1
        model += (
            s[i] + pulp.lpSum(b[(i, j)] * j_i[i, j] for j in ORIENTATIONS)
            <= s[i2] + TOLERANCE + M * (2 - x[i, k] - x[i2, k + 1])
        )
# 5.
for i in TOYS:
    model += s[i] + pulp.lpSum(b[(i, j)] * j_i[i, j] for j in ORIENTATIONS) <= rightmost

# === Objective Function ===
model += rightmost

# === Solve ===
solver = pulp.PULP_CBC_CMD(msg=False)
model.solve(solver)

# === Collect toy data by position ===
arrangement = {}

for i in TOYS:
    pos = [k for k in POSITIONS if pulp.value(x[i, k]) > 0.5][0]
    ori = [j for j in ORIENTATIONS if pulp.value(j_i[i, j]) > 0.5][0]
    arrangement[pos] = f"{i}{arrow[ori]}"

# === Print toys in order of position ===
ordered_output = " ".join(arrangement[k] for k in sorted(arrangement))
print(ordered_output)

# === Calculate total length ===
starts = {}
ends = {}

for i in TOYS:
    si = pulp.value(s[i])
    ori = [j for j in ORIENTATIONS if pulp.value(j_i[i, j]) > 0.5][0]
    length = b[(i, ori)]
    starts[i] = si
    ends[i] = si + length

total_length = max(ends.values()) - min(starts.values())
print(f"Total arrangement length: {total_length:.2f} cm")
```

Listing 1. Overlap Minimisation Model for Four Strongest Puzzle

Appendix B
Data CSV File

animal	orientation	value
dragon	up	10.576
tiger	up	7.400
lion	up	6.633
garuda	up	7.200
dragon	down	10.576
tiger	down	7.400
lion	down	6.633
garuda	down	7.200

Appendix C
AI Prompts

Various prompts were used throughout this project; from code help and explanation to brainstorming and formatting LaTeX.

List of AI Prompts

```
>> Here's an extract from the brief of my maths optimisation coursework.
>> Let's brainstorm some ideas for an alternative formulation.
>> Let's dive further into this option.
```

```
>> How can I solve this formulation in Python? What library should I use?
>> Here's the measurements in a CSV file. How can I read the file and use the
   measurements in my Python code?
>> Let's go over the variables, constraints and objective function again. How can
   I format them for latex?
>> What is the Big-M method? Is it like a way to have conditional constraints?
>> How can I change the output of the solver so that animals are printed in the
   order they appear (that is, in increasing k)?
>> I'd like to have up and down arrow unicode instead of the words up and down.
>> How can I include Python code in the appendix of my report? My report is
   two-column, so bear that in mind.
>> Let's allow small overlaps up to a certain threshold. Does constraint 4 change
   at all?
>> Can you explain constraint 5 again?
>> Here's my report so far. Can you compare it against the brief. What have I done
   well? What still needs improvements? What haven't I covered yet?
>> How do I format a table in latex to summarise the symbols I've used for my
   formulation so far?
>> How can I make text wrap so that it doesn't spill off the right side of the
   page?
```