A Non-Linear Synergy Model for Fairy Chess Piece Valuation

Zekk[*]

August 27, 2025

Abstract

Traditional chess piece valuation relies on a simple additive model, which fails to account for the synergistic interactions between pieces. This paper introduces a novel, non-linear composition law to model these synergies, defined as $v(A \oplus B) = v(A) + v(B) + c \cdot v(A) \cdot v(B)$, where c is a constant representing the intensity of the synergy. We provide theoretical justification for our model, showing it to be mathematically sound and not merely an arbitrary second-order approximation.

We determine the base values for a set of 10 primitive move/capture types and the synergy constant c by optimizing them to fit a set of 20 empirically derived target values for compound pieces. The optimization was performed using a differential evolution algorithm, and the objective was to minimize the maximum relative error (a minimax approach) to ensure a reliable and balanced model.

The resulting model, with 11 core parameters, was rigorously validated against unseen data, showing no signs of overfitting. We further validated the model's predictions against established, engine-derived values for pieces in several well-known chess variants (Capablanca Chess, Orda Chess, Empire Chess), finding a remarkable correlation. The final, validated model provides a comprehensive and internally consistent lexicon of values for over 280 unique fairy chess pieces, offering a framework for quantitative analysis in the domain of chess variants.

Contents

[*]The mathematical framework and core methodology of this paper were developed by the author. Google's Gemini model was utilized as a tool for assistance in generating and debugging the Python and LaTeX code. The author directed this process and assumes full responsibility for the final verification, correctness, and interpretation of all results presented.

1 Methodology

1.1 The Synergy Composition Law

We propose that the value of a compound piece, formed by the combination of two independent components A and B, can be described by the following composition law, which we denote with the $\star$ operator:

$$v(A \oplus B) = v(A) \star v(B) = v(A) + v(B) + c \cdot v(A) \cdot v(B) \tag{1}$$

Here, $v(A)$ and $v(B)$ are the values of the individual components, and c is a small, positive constant representing the "synergy coefficient" of the system. This law is associative and commutative, allowing for the composition of any number of components. The term $c \cdot v(A) \cdot v(B)$ represents the synergy bonus—the idea that powerful pieces are better at amplifying each other's strengths. In Subsection 1.5, we demonstrate that this composition law is not merely a second-order approximation of the problem: higher-order terms are likely to be zero.

1.2 The Optimization Problem

Our model is defined by 11 fundamental parameters, which form our parameter vector $\mathbf{p}$:

- The synergy constant, c.
- 5 parameters for the value of "moving as" a primitive piece: $v(\text{move\_as\_X})$ for X in {Ferz, Wazir, Bishop, Knight, Rook}.
- 5 parameters for the value of "capturing as" a primitive piece: $v(\text{capture\_as\_X})$ for X in {Ferz, Wazir, Bishop, Knight, Rook}.

The final value of a piece is the composition of its total move value and its total capture value, i.e., $v(\text{Piece}) = v(\text{MoveSet}) \star v(\text{CaptureSet})$.

To find the optimal values for $\mathbf{p}$, we minimize a loss function $\mathcal{L}$ defined by the difference between our model's predictions and a set of 20 empirically derived target values, T_i, for various compound pieces. We chose to minimize the maximum relative error to ensure model reliability and avoid large errors on any single piece:

$$\min_{\mathbf{p}} \mathcal{L}(\mathbf{p}) = \min_{\mathbf{p}} \left(\max_{i=1}^{20} \left| \frac{f(\mathbf{p})_i - T_i}{T_i} \right| \right) \tag{2}$$

where $f(\mathbf{p})_i$ is the value of the i-th compound piece as calculated by our model using the parameters $\mathbf{p}$.

1.3 Optimization and Validation Process

The optimization was performed using the `differential_evolution` algorithm from the SciPy library. The process was iterative and involved several key stages:

1. **Initial Optimization:** We began by optimizing the 11 parameters against the 20 target values using a mean-relative-error objective.

2. **Model Validation (Overfitting/Sanity Test):** To ensure the model was not merely fitting to the data, we performed a validation test. We held out 3 of the 20 constraints as a "test set" and re-optimized the model on the remaining 17. The model's performance on the unseen test data was nearly identical to its performance on the training data, providing moderate evidence that the model generalizes well and is not overfit.

3. **Objective Refinement (Minimax):** Analysis of the mean-optimized model revealed that while the average error was low, a few constraints had a disproportionately high error. To create a more reliable and balanced model, we changed the objective function to minimize the maximum relative error, as described above.

4. **Final Model Training:** Having validated the model's structure and chosen the minimax objective, we performed a final, comprehensive optimization run using the complete set of 20 constraints to produce the definitive set of parameters.

1.4 Source of Target Values

The 20 target values (T_i) used as the ground truth for our optimization are not arbitrary. They are derived from the results of a large-scale empirical study detailed in the paper "Empirical values of fairy chess pieces"[1].

The methodology of that study involved:

1. **Data Generation:** Using a modified version of the Fairy-Stockfish engine to play over 60 millions matches between different custom armies, generating a dataset of over 600 million unique positions. The variant used standard chess rules on an 8x8 board but allowed for non-standard starting armies composed of various fairy pieces.

2. **Value Estimation:** A logistic regression model was fitted to this massive dataset to estimate the practical value of each piece based on its contribution to the game outcome (win/loss/draw).

3. **Normalization:** The resulting values were normalized by fixing the value of a Pawn to 1.00. For use in our model, we have rescaled these values by a factor of 100 (e.g., a Queen value of 8.50 in the paper becomes a target of 850 in our optimization).

This empirical data provides a robust, engine-verified baseline for the practical strength of compound pieces. Our goal is to determine if our theoretical synergy law can independently discover a set of fundamental parameters that accurately reproduces these empirically observed values. The 20 specific constraints used in our final optimization (see Table 2) correspond directly to the values derived from this first empirical method. For example, the paper's value of 7.62 for the Chancellor (`NR`) corresponds to our target value of 762.

[1]see `https://github.com/Zekkzekk/Fairy-chess/blob/main/Empirical_values_of_fairy_chess_pieces.pdf`

1.5 Mathematical Foundation of the Model

The choice of the synergy law in Equation 1 is not arbitrary but is grounded in the mathematical theory of formal group laws. Here, we briefly outline the reasoning that leads to this specific functional form.

Let us model the set of all possible primitive piece powers as a basis for a finite-dimensional real vector space, E. A fairy chess piece $A \in E$ can be represented by its vector of capabilities (e.g., coordinates for sliding North, leaping in a (2,1) pattern, etc.). Two pieces, A and B, are considered *independent* if their powers do not overlap, which can be formalized by stating that the term-wise (Hadamard) product of their coordinate vectors is zero. The combination of two such independent pieces, $A \oplus B$, is represented by vector addition in E.

The value of a piece is a map $v : E \to \mathbb{R}$. We seek to find a function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that for any two independent pieces A and B:

$$v(A \oplus B) = f(v(A), v(B)) \tag{3}$$

It is natural to assume that this function $f(x, y)$ is analytic and can therefore be expressed as a power series over $\mathbb{R}$:

$$f(x,y) = \sum_{i,j\geq 0} \alpha_{i,j} x^i y^j \quad \text{where } \alpha_{i,j} \in \mathbb{R} \tag{4}$$

This function must satisfy two fundamental properties derived from the logic of chess piece combinations:

1. **Existence of an Identity Element:** There exists a "null piece" $0_E \in E$ which has no move or capture abilities. It is natural to assume this piece has zero value, i.e., $v(0_E) = 0$.[2] Combining any piece A with the null piece does not change its capabilities, so $v(A \oplus 0_E) = v(A)$. This implies:

   $$f(x, 0) = x \quad \text{and by commutativity,} \quad f(0, y) = y \tag{5}$$

   This identity requirement forces the power series to be of the form $f(x, y) = x + y + \sum_{i,j\geq 1} \alpha_{i,j} x^i y^j$.

2. **Associativity:** The order of composition should not matter. For example, a Centaur can be seen as (Ferz + Wazir) + Knight, or as Ferz + (Wazir + Knight). This physical reality demands that the value function must be associative:

   $$f(f(x, y), z) = f(x, f(y, z)) \tag{6}$$

A power series satisfying these two properties is, by definition, a **1-dimensional commutative formal group law** over the ring $\mathbb{R}$. While the theory of formal group laws is deep, a crucial result applies to our context. For our real-world application, it is reasonable to assume that the complex interactions do not require an infinite series and can be modeled by a polynomial. If we assume $f(x, y)$ is a polynomial, a well-known theorem in the theory of formal group laws states that any polynomial formal group law over a ring of characteristic zero (such as $\mathbb{R}$) must be of the form:

$$f(x, y) = x + y + c \cdot xy \tag{7}$$

for some constant c in the ring.

Thus, the seemingly simple choice for our synergy law is, in fact, the only possible polynomial function that satisfies the fundamental and logically necessary property of associativity. This provides a strong theoretical justification for our model, showing it to be mathematically sound and not merely an arbitrary approximation.

[2] One could argue a "wall" of null pieces might have a non-zero value (either positive for defense or negative for obstruction). However, for the composition of active powers, a null piece must act as a neutral identity.

2 Results and Discussion

2.1 The Definitive Parameter Set

The final optimization run yielded the following stable and robust set of 11 parameters. These values form the foundation for all subsequent calculations(see Table 1).

Table 1: Final Optimized Model Parameters.

Parameter	Final Optimized Value
Synergy Constant (c)	0.000513
`move_as_ferz`	53.235816
`move_as_wazir`	27.130748
`move_as_bishop`	64.983722
`move_as_knight`	89.486176
`move_as_rook`	112.491106
`capture_as_ferz`	100.921852
`capture_as_wazir`	103.121038
`capture_as_bishop`	247.783480
`capture_as_knight`	188.077510
`capture_as_rook`	296.090429

The minimax optimization resulted in a final maximum relative error of **4.15%** across all 20 constraints, with a mean relative error of **3.00%** (see Table 2).

Interpretation of Parameters

An analysis of the optimized parameters in Table 1 reveals several insightful properties of the system. A fundamental imbalance exists between movement and capture values, with the `capture_as_X` parameters consistently being larger than their `move_as_X` counterparts. This suggests that in the complex, often crowded positions typical of fairy chess, a piece's offensive reach and direct threat potential (*capture ability*) contributes more to its overall value than its pure mobility.

While an initial analysis suggested a rough linear approximation of capture_value $\approx 2.5 \times$ move_value, the final parameters reveal a more complex, non-monotonic relationship. For instance, the model finds that move_as_ferz $>$ move_as_wazir, reflecting the common human observation that the diagonal-moving Ferz is "faster" at traversing the board than the orthogonal-moving Wazir. However, the model simultaneously finds that capture_as_ferz $<$ capture_as_wazir, aligning with the intuition that the Wazir is a better attacking piece.

A similar inversion occurs between the Bishop and Knight. The model finds move_as_bishop $<$ move_as_knight, yet capture_as_bishop $>$ capture_as_knight. This elegantly quantifies a well-known strategic principle: the Knight's "leaping" ability makes it an excellent mobile piece, especially in the opening and middle game phases, but the Bishop's long-range sliding nature gives it a higher raw attacking and checkmating potential.

The fact that these parameters are interpretable and align with established strategic concepts is a significant advantage of this modeling approach. While a machine learning model, such as an NNUE (Efficiently Updatable Neural Network), could likely achieve a lower error by training on the data, its internal parameters would function as a "black box." In contrast, our model provides a clear and conceptually simple framework that offers explanatory power in addition to its predictive accuracy.

Table 2: Model Performance on the 20 Target Constraints.

Constraint Name	Model Value	Target Value	Rel. Error (%)
B+CB+R+CR	816.7897	850.0000	3.9071
N+CN+R+CR	774.3548	762.0000	1.6214
B+CB+N+CN	654.3528	679.0000	3.6299
F+W+N+CF+CW+CN	629.3217	605.0000	4.0201
F+R+CF+CR	616.8425	598.0000	3.1509
B+W+CB+CW	474.3990	489.0000	2.9859
R+CR	425.6669	426.0000	0.0782
B+CR	370.9440	387.0000	4.1488
N+CR	399.1679	384.0000	3.9500
F+W+CR	389.5166	374.0000	4.1488
R+CB	374.5725	364.0000	2.9045
R+CF+CW	333.9544	328.0000	1.8154
B+CB	321.0268	328.0000	2.1260
F+W+CB	339.1999	326.0000	4.0490
R+CN	311.4213	307.0000	1.4402
N+CF+CW	308.4787	304.0000	1.4732
N+CN	286.1969	297.0000	3.6374
F+W+CN	277.0099	289.0000	4.1488
B+CF+CW	281.3446	289.0000	2.6489
B+CN	259.3306	249.0000	4.1488

2.2 Comparison with External Data (Fairy-Stockfish)

To ground our theoretical model in an empirical context, we compared its predictions to engine-derived values from Fairy-Stockfish, a powerful chess variant engine. We selected well-known variants featuring pieces whose powers could be constructed from our model's primitive components.

Normalization Methodology

Engine-based piece values are on a different scale from our own. To create a direct comparison, we normalized the Fairy-Stockfish values. Both our model and the engine use the Knight as a common, well-defined piece. We therefore scale all engine values by a factor that equates its Knight value to our model's Knight value.

Based on the Fairy-Stockfish "Early Game" value for a Knight (781) and our model's final optimized value (286.20), the normalization factor is:

$$\text{Normalization Factor} = \frac{\text{Our Model's Knight Value}}{\text{SF Knight Value}} = \frac{286.1969}{781} \approx 0.36645 \qquad (8)$$

All Fairy-Stockfish values cited below have been scaled by this factor.

A Note on Fairy-Stockfish's Evaluation

It is important to understand that modern versions of Fairy-Stockfish do not use a simple table of hardcoded values for compound pieces like the Archbishop or Chancellor. Instead, it employs a sophisticated programmatic approach. The engine's `piece_value` function (located in `psqt.cpp`) calculates a piece's value from first principles, based on a weighted sum of its fundamental abilities:

- The number of non-sliding "leaping" moves (for capture and quiet moves).
- The number and range of "sliding" moves (for capture and quiet moves).
- The number of "hopping" moves.
- Bonuses for orthogonal vs. diagonal sliding capabilities.

This weighted sum is then passed through a non-linear scaling function, $v' = v \cdot e^{v/10000}$, to arrive at the final value. Therefore, the values we use for comparison, such as 1735 for the Archbishop, are the *emergent results* of this complex calculation, not predefined constants. This makes the close agreement with our model even more remarkable.

Comparison of Methodologies

Our model and Fairy-Stockfish represent two fundamentally different but complementary philosophies for piece valuation.

Fairy-Stockfish's Model is a **bottom-up, feature-based system**. It begins with the most primitive aspects of movement (a single step, a slide) and builds piece values from a weighted sum of these features. Its goal is to create the strongest possible playing entity. The weights are not designed to be human-interpretable but are tuned automatically over millions of games to maximize playing performance. It is highly contextual, with its base values being just the first step in a deeper evaluation that includes complex piece-square tables (PSTs).

Our Model is a **top-down, theoretical system**. It begins with the high-level, empirically-observed values of powerful *compound* pieces and seeks to find a simple, universal law that explains them. Its goal is to create a human-interpretable and elegant theory of piece synergy. The core of the model is not a collection of feature weights, but the single, explicit synergy law: $v(A+B) = v(A) + v(B) + c \cdot v(A) \cdot v(B)$. While our model produces a single value for each piece, the framework could easily be extended to derive contextual PSTs, provided reliable target values for each square were available.

The key difference is one of purpose: Fairy-Stockfish builds a complex, performant engine; our model builds a simple, explanatory theory. The following comparisons show how closely these two different philosophies converge.

2.2.1 Variant 1: Capablanca Chess

Capablanca Chess introduces two major compound pieces that serve as excellent test cases for our synergy law.

- **The Archbishop (`BN`):** Combines the powers of a Bishop and a Knight.
- **The Chancellor (`NR`):** Combines the powers of a Rook and a Knight.

Table 3: Comparison with Capablanca Chess Piece Values.

Piece	Model Value	SF Raw Val.	SF Norm. Val.	Diff. (%)
Chancellor	774.43	1960.00	718.24	7.74
Archbishop	654.40	1735.00	635.81	2.92

The model performs exceptionally well. It correctly predicts that the Chancellor is significantly stronger than the Archbishop. The quantitative agreement is also strong, with the Archbishop's value being predicted to within 3% of the engine's value.

2.2.2 Variant 2: Empire Chess

Empire Chess features four powerful divergent pieces, where the move-set differs from the capture-set. This provides a robust test of our model's ability to handle this distinction.

- **The Cardinal (`quebis`):** Moves like a Queen (`BR`), but captures like a Bishop (`B`).
- **The Tower (`queroo`):** Moves like a Queen (`BR`), but captures like a Rook (`R`).
- **The Duke (`queman`):** Moves like a Queen (`BR`), but captures like a King/Man (`FW`).
- **The Eagle (`quekni`):** Moves like a Queen (`BR`), but captures like a Knight (`N`).

Table 4: Comparison with Empire Chess Piece Values.

Piece	Model Value	SF Raw Val.	SF Norm. Val.	Diff. (%)
Tower	504.83	1375.00	503.87	0.15
Cardinal	452.00	1225.00	448.90	0.69
Duke	410.06	1050.00	384.77	6.57
Eagle	386.80	1000.00	366.45	5.55

Again, the correlation is remarkably strong. The model correctly predicts the exact relative ranking of all four pieces. For the two pieces that retain a long-range capture (Tower and Cardinal), the model's prediction is nearly perfect, with less than 1% error. For the pieces that combine long-range movement with short-range captures (Duke and Eagle), the model values them slightly higher than the engine, suggesting a potential area where the universal synergy law differs from an engine's situational, empirical evaluation.

2.2.3 Variant 3: Orda Chess

Orda Chess presents a unique set of pieces, providing a strong test for the model's ability to evaluate both compound pieces (combining move types) and divergent pieces (different move and capture).

- **The Kheshig (`FNW`):** Defined as moving and capturing like a Knight and a King combined. In our model, this is the **Centaur**.
- **The Lancer (`kniroo`):** A divergent piece that moves like a Knight (`N`), but captures like a Rook (`R`).
- **The Horse Archer (`knibis`):** A divergent piece that moves like a Knight (`N`), but captures like a Bishop (`B`).
- **The Yurt (`F`):** Moves like a Shogi "Silver General" (one step forward or one step diagonally). As our model's primitives do not include a "forward-only" component, we use the closest available piece, the **Ferz**[3], which moves one step diagonally.

The results of this comparison are highly informative. The model performs exceptionally well on the two most powerful and complex pieces: its valuation for the Kheshig (Centaur) and the Lancer are both within 5% of the engine-derived values. This is a strong validation of the synergy law's ability to handle both compound and divergent pieces.

[3]This is a known simplification. The forward-move capability of the Silver General likely accounts for a significant portion of the value discrepancy observed for the Yurt.

Table 5: Comparison with Orda Chess Piece Values.

Piece	Model Value	SF Raw Val.	SF Norm. Val.	Diff. (%)
Kheshig	628.79	1800.00	659.61	−4.67
Lancer	397.31	1050.00	384.77	3.26
Horse Archer	343.36	1100.00	403.10	−14.82
Yurt	156.91	630.00	230.86	−32.03

The discrepancies are concentrated in the two weaker pieces. The large error for the Yurt is anticipated and can be attributed to the necessary simplification of its moveset in our model. The notable difference for the Horse Archer continues a pattern where our model's universal synergy law appears to value the combination of a short-range move (Knight) with a long-range capture (Bishop) differently than the engine's empirical, game-based evaluation.

2.3 A Comprehensive Lexicon of Fairy Piece Values

The primary output of this work is a comprehensive and self-consistent ranking of fairy chess pieces. The validated model was used to programmatically generate all valid, non-redundant compound pieces. A representative sample is shown in Table 6. The full list is available in Appendix C.

Table 6: A Sample from the Comprehensive Piece Value Lexicon.

Betza	Name	Value
BNR	Amazon (Rook+Bishop+Knight)	1223.03
FNR	Ferz+Knight+Rook	994.59
BR	Queen (Rook+Bishop)	816.92
NR	Chancellor (Rook+Knight)	774.43
BN	Archbishop (Bishop+Knight)	654.40
FNW	Centaur (Ferz+Wazir+Knight)	628.79
FR	Berserker (Ferz+Rook)	617.95
BW	Dragon Horse (Bishop+Wazir)	470.16
FN	Ferz+Knight	464.73
R	Rook	425.67
B	Bishop	321.02
FW	Man (King) (Ferz+Wazir)	298.54
N	Knight	286.20
F	Ferz	156.91
W	Wazir	132.25

3 Conclusion and Future Work

We have successfully developed and validated a novel non-linear model for fairy chess piece valuation. The proposed synergy law provides a powerful framework for quantifying the value of compound pieces. The derived parameter set is robust, predictive, and aligns closely with external, engine-based evaluations. This work demonstrates that the complex interactions between chess pieces can be effectively modeled by a simple, universal law of composition. The resulting piece value lexicon offers a valuable tool for analysts and designers of chess variants.

3.1 Large-Scale Empirical Validation

The current model was optimized using 20 empirically-derived target values. While our validation tests show that the model generalizes well, its ultimate test lies in its ability to predict the values of the many other pieces it can describe. Our framework can generate values for approximately 288 unique, non-redundant compound pieces.

A crucial next step would be to generate a larger set of empirical target values to test these predictions on a wider scale. A robust methodology for this would be:

1. Train a modern, strong chess variant engine (e.g., an NNUE-based version of Fairy-Stockfish) on a variant that includes a much larger set of the compound pieces defined in our lexicon.
2. Use this trained engine to generate a new, massive dataset of several billion positions.
3. Apply the same logistic regression methodology used in the source paper for our initial targets to derive new, independent empirical values for this larger set of pieces.

Comparing the resulting values against our model's a priori predictions would serve as a comprehensive validation of the synergy law's predictive power and help identify its precise limitations.

3.2 Decomposition into Atomic Directional Components

Our current model uses five primitive pieces (Ferz, Wazir, Knight, Bishop, Rook) as its fundamental building blocks. A deeper theoretical model could be developed by decomposing these primitives into even more elementary "atomic" components based on directional symmetry.

This requires defining an inverse operation for our composition law. For any positive value $a \in \mathbb{R}$, we can define its "star square root", denoted $\sqrt[\star]{(a)}$, as the unique positive real number x such that $x \star x = a$. This x is the positive root of the quadratic equation $cx^2 + 2x - a = 0$:

$$\sqrt[\star]{(a)} = x = \frac{-1 + \sqrt{1 + ca}}{c} \tag{9}$$

Using this operator, we can decompose our primitives based on the symmetry of their moves:

- **Wazir and Ferz:** The Wazir's move can be seen as the composition of four independent, symmetric orthogonal steps (e.g., '(0,1)'). The value of a single orthogonal step, $v(\text{step}_{\text{orth}})$, could be found by repeated application of the star square root: $v(\text{step}_{\text{orth}}) = \sqrt[\star]{(\sqrt[\star]{(v(\text{Wazir}))})}$. A similar decomposition applies to the four diagonal steps of the Ferz.
- **Knight:** The Knight's move consists of eight symmetric leaps (e.g., '(2,1)'). The value of a single leap could be defined as $v(\text{leap}_{2,1}) = \sqrt[\star]{(\sqrt[\star]{(\sqrt[\star]{(v(\text{Knight}))})})}$.

This deeper model, based on atomic directional components, would allow for the construction of a much wider range of fairy pieces, including those with asymmetric movesets like the Shogi Silver General (Yurt). A significant challenge to this approach, however, is that it assumes perfect symmetry of the board, an assumption that is often violated in practice due to factors like the kings' positions and the inherent asymmetries of the opening and middlegame. Validating such a model would require extremely nuanced empirical data.

A Python Script for Optimization process

```python
import numpy as np
from scipy.optimize import differential_evolution

# Global counter for iterations to use in the callback function
iteration_counter = 0

def objective_function(params):
"""
This is the final objective function. It uses all 20 constraints
from the original dataset and minimizes the MAXIMUM relative error.
"""
# Unpack the 11 variables from the input array
cst, move_as_ferz, move_as_wazir, move_as_bishop, move_as_knight, \
move_as_rook, capture_as_ferz, capture_as_wazir, capture_as_bishop, \
capture_as_knight, capture_as_rook = params

def star(*args):
"""
Applies the associative law x * y = x + y + cst*x*y to a list of arguments.
"""
result = args[0]
for i in range(1, len(args)):
result = result + args[i] + cst * result * args[i]
return result

# --- MODIFICATION: Using all 20 original constraints ---
constraints = [
# The 3 "test" constraints are now included in the optimization
abs(star(move_as_bishop, capture_as_bishop, move_as_rook, capture_as_rook) - 850) / 850,
abs(star(move_as_knight, capture_as_knight, move_as_rook, capture_as_rook) - 762) / 762,
abs(star(move_as_bishop, capture_as_bishop, move_as_knight, capture_as_knight) - 679) / 679,
# The original 17 "training" constraints
abs(star(move_as_ferz, move_as_wazir, move_as_knight, capture_as_ferz, capture_as_wazir, capture_as_knight) - 605) / 605,
abs(star(move_as_ferz, move_as_rook, capture_as_ferz, capture_as_rook) - 598) / 598,
abs(star(move_as_bishop, move_as_wazir, capture_as_bishop, capture_as_wazir) - 489) / 489,
abs(star(move_as_rook, capture_as_rook) - 426) / 426,
abs(star(move_as_bishop, capture_as_rook) - 387) / 387,
abs(star(move_as_knight, capture_as_rook) - 384) / 384,
abs(star(move_as_ferz, move_as_wazir, capture_as_rook) - 374) / 374,
abs(star(move_as_rook, capture_as_bishop) - 364) / 364,
abs(star(move_as_rook, capture_as_ferz,
   ↪ capture_as_wazir) - 328) / 328,
abs(star(move_as_bishop, capture_as_bishop) - 328) /
   ↪ 328,
abs(star(move_as_ferz, move_as_wazir, capture_as_bishop
   ↪ ) - 326) / 326,
abs(star(move_as_rook, capture_as_knight) - 307) / 307,
abs(star(move_as_knight, capture_as_ferz,
   ↪ capture_as_wazir) - 304) / 304,
abs(star(move_as_knight, capture_as_knight) - 297) /
   ↪ 297,
abs(star(move_as_ferz, move_as_wazir, capture_as_knight
   ↪ ) - 289) / 289,
abs(star(move_as_bishop, capture_as_ferz,
   ↪ capture_as_wazir) - 289) / 289,
abs(star(move_as_bishop, capture_as_knight) - 249) /
   ↪ 249
]
# --- END MODIFICATION ---

# Optimize for the worst-case (maximum) error
return np.max(constraints)

def callback_function(xk, **kwargs):
"""
Callback function to print intermediate results.
"""
global iteration_counter
iteration_counter += 1
if iteration_counter % 20 == 0:
current_value = objective_function(xk)
print(f"Iteration: {iteration_counter}, Minimized MAX
   ↪ Value: {current_value:.8f}")
print("Current Parameters:")
np.set_printoptions(formatter={'float': '{: 10.6f}'.
   ↪ format})
print(xk)
print("-" * 50)

# --- Main execution guarded for multiprocessing safety
   ↪ ---
if __name__ == "__main__":
# 1. Define the bounds for the 11 unknown variables
bounds = [(1e-6, 1)] + [(0, 500)] * 10

# 2. Provide a starting array using the best result
   ↪ from the previous minimax run
initial_guess = np.array([
0.000513, 53.235055, 27.130596, 64.982995, 89.486176,
   ↪ 112.521226,
100.921915, 103.528723, 247.760112, 188.077800,
   ↪ 296.090446
])

# Create an initial population
num_population_members = 6
```

```
initial_population = np.zeros((num_population_members,
    ↪ len(initial_guess)))
initial_population[0] = initial_guess

for i in range(1, num_population_members):
cst_val = np.random.uniform(bounds[0][0], bounds[0][1])
other_vals = np.random.uniform(bounds[1][0], bounds
    ↪ [1][1], 10)
initial_population[i] = np.hstack(([cst_val],
    ↪ other_vals))

# 3. Run the differential evolution algorithm
result = differential_evolution(
objective_function,
bounds,
init=initial_population,
callback=callback_function,
disp=True,
maxiter=10000,
seed=42,
tol=1e-20,
atol=1e-208,
popsize=7500000,
workers=6
)

# 4. Print the final results
print("\n" + "="*65)
print("Optimization finished (Minimizing MAXIMUM Error
    ↪ on ALL 20 Targets).")
print(f"Final minimized MAXIMUM relative error: {result
    ↪ .fun:.6%}")
print("\nOptimal parameters found:")

param_names = [
"cst", "move_as_ferz", "move_as_wazir", "move_as_bishop
    ↪ ",
"move_as_knight", "move_as_rook", "capture_as_ferz",
"capture_as_wazir", "capture_as_bishop", "
    ↪ capture_as_knight",
"capture_as_rook"
]

for name, value in zip(param_names, result.x):
print(f"{name:<20}: {value:10.6f}")

# --- SETUP FOR FINAL REPORT ---
cst, move_as_ferz, move_as_wazir, move_as_bishop,
    ↪ move_as_knight, \
move_as_rook, capture_as_ferz, capture_as_wazir,
    ↪ capture_as_bishop, \
capture_as_knight, capture_as_rook = result.x

def star(*args):
res = args[0]
for i in range(1, len(args)):
res = res + args[i] + cst * res * args[i]
return res
```

```python

# --- DETAILED CONSTRAINT REPORT (ALL 20 CONSTRAINTS) ---
print("\n" + "-"*65)
print("Final Report (All 20 Constraints):")
print("-" * 65)

all_constraints = [
("B+CB+R+CR", star(move_as_bishop, capture_as_bishop, move_as_rook, capture_as_rook), 850),
("N+CN+R+CR", star(move_as_knight, capture_as_knight, move_as_rook, capture_as_rook), 762),
("B+CB+N+CN", star(move_as_bishop, capture_as_bishop, move_as_knight, capture_as_knight), 679),
("F+W+N+CF+CW+CN", star(move_as_ferz, move_as_wazir, move_as_knight, capture_as_ferz, capture_as_wazir, capture_as_knight), 605),
("F+R+CF+CR", star(move_as_ferz, move_as_rook, capture_as_ferz, capture_as_rook), 598),
("B+W+CB+CW", star(move_as_bishop, move_as_wazir, capture_as_bishop, capture_as_wazir), 489),
("R+CR", star(move_as_rook, capture_as_rook), 426),
("B+CR", star(move_as_bishop, capture_as_rook), 387),
("N+CR", star(move_as_knight, capture_as_rook), 384),
("F+W+CR", star(move_as_ferz, move_as_wazir, capture_as_rook), 374),
("R+CB", star(move_as_rook, capture_as_bishop), 364),
("R+CF+CW", star(move_as_rook, capture_as_ferz, capture_as_wazir), 328),
("B+CB", star(move_as_bishop, capture_as_bishop), 328),
("F+W+CB", star(move_as_ferz, move_as_wazir, capture_as_bishop), 326),
("R+CN", star(move_as_rook, capture_as_knight), 307),
("N+CF+CW", star(move_as_knight, capture_as_ferz, capture_as_wazir), 304),
("N+CN", star(move_as_knight, capture_as_knight), 297),
("F+W+CN", star(move_as_ferz, move_as_wazir, capture_as_knight), 289),
("B+CF+CW", star(move_as_bishop, capture_as_ferz, capture_as_wazir), 289),
("B+CN", star(move_as_bishop, capture_as_knight), 249)
]

print(f"{'Constraint Name':<18} | {'Calculated':>12} | {'Target':>10} | {'Rel. Error (%)':>15}")
print(f"{'-'*18: <18} | {'-'*12:>12} | {'-'*10:>10} | {'-'*15:>15}")

all_errors = []
for name, calc_val, target_val in all_constraints:
rel_error = (abs(calc_val - target_val) / target_val) * 100
all_errors.append(rel_error)
print(f"{name:<18} | {calc_val:12.4f} | {target_val:10} | {rel_error:14.4f}%")

print("-" * 65)
print(f"AVERAGE error on all 20 constraints (for comparison): {np.mean(all_errors):.4f}%")
print(f"MAXIMUM error on all 20 constraints (optimized value): {np.max(all_errors):.4f}%")
print("="*65)
```

Listing 1: The complete Python script used to optimize the parameters

B Final Python Script for Piece Value Calculation

```python
import pandas as pd
from itertools import combinations, product

class FairyPiece:
def __init__(self, name, betza_components, move_val, capture_val):
self.name = name
self.betza_components = set(betza_components)
self.betza = "".join(sorted(list(self.betza_components)))
self.move_value = move_val
self.capture_value = capture_val

def calculate_all_piece_values():
params = {
        "cst": 0.000513,
        "move_as_ferz": 53.235816, "capture_as_ferz": 100.921852,
        "move_as_wazir": 27.130748, "capture_as_wazir":  103.121038,
        "move_as_knight": 89.486176, "capture_as_knight": 188.077510,
        "move_as_bishop": 64.983722, "capture_as_bishop": 247.783480,
        "move_as_rook": 112.491106, "capture_as_rook": 296.090429,
}
cst = params["cst"]

def star(*args):
result = args[0]
for i in range(1, len(args)):
result = result + args[i] + cst * result * args[i]
return result

base_pieces_map = {
        "F": FairyPiece("Ferz", "F", params["move_as_ferz"], params["capture_as_ferz"]),
        "W": FairyPiece("Wazir", "W", params["move_as_wazir"], params["capture_as_wazir"]),
        "N": FairyPiece("Knight", "N", params["move_as_knight"], params["capture_as_knight"]),
        "B": FairyPiece("Bishop", "B", params["move_as_bishop"], params["
    capture_as_bishop"]),
    "R": FairyPiece("Rook", "R", params["move_as_rook"], params["capture_as_rook"]),
}

canonical_definitions = {
    "F": ("Ferz", "fer"), "W": ("Wazir", "waz"), "N": ("Knight", "kni"),
    "B": ("Bishop", "bis"), "R": ("Rook", "roo"), "FW": ("Man (King)", "man"),
    "BR": ("Queen", "que"), "NR": ("Chancellor", "cha"), "BN": ("Archbishop", "arc"),
    "BNR": ("Amazon", "ama"), "FNW": ("Centaur", "cen"), "FR": ("Berserker", "ber"),
    "BW": ("Dragon Horse", "dra"), "FN": ("Ferz+Knight", "fn"),
    "NW": ("Knight+Wazir", "nw"), "BNW": ("Bishop+Knight+Wazir", "bnw"),
    "FNR": ("Ferz+Knight+Rook", "fnr"),
}

standard_pieces = {}
base_betza_keys = list(base_pieces_map.keys())

for i in range(1, len(base_betza_keys) + 1):
for combo_keys in combinations(base_betza_keys, i):
component_set = set(combo_keys)

if 'B' in component_set and 'F' in component_set: continue
if 'R' in component_set and 'W' in component_set: continue

combo_pieces = [base_pieces_map[key] for key in combo_keys]
new_betza = "".join(sorted(list(component_set)))

if new_betza in canonical_definitions:
new_move_val = star(*[p.move_value for p in combo_pieces])
new_capture_val = star(*[p.capture_value for p in combo_pieces])
new_name = canonical_definitions[new_betza][0]
standard_pieces[new_betza] = FairyPiece(new_name, component_set, new_move_val, new_capture_val)

final_results = []
abbreviation_map = {v[0]: v[1] for k, v in canonical_definitions.items()}
valid_standard_pieces = list(standard_pieces.values())

for move_piece in valid_standard_pieces:
for capture_piece in valid_standard_pieces:
if move_piece.betza == capture_piece.betza:
final_value = star(move_piece.move_value, move_piece.capture_value)
```

```
                    final_results.append({ "Betza": move_piece.betza, "Name
                       ↪ ": move_piece.name, "Value": final_value})
                    else:
                    move_abbr = abbreviation_map.get(move_piece.name, "xxx"
                       ↪ )
                    capture_abbr = abbreviation_map.get(capture_piece.name,
                       ↪  "yyy")
                    betza_code = f"{move_abbr}{capture_abbr}"
                    descriptive_name = f"Moves like {move_piece.name},
                       ↪ Captures like {capture_piece.name}"
                    final_value = star(move_piece.move_value, capture_piece
                       ↪ .capture_value)
                    final_results.append({"Betza": betza_code, "Name":
                       ↪ descriptive_name, "Value": final_value})

                    df = pd.DataFrame(final_results).drop_duplicates(subset
                       ↪ =['Name'])
                    df_sorted = df.sort_values(by="Value", ascending=False)
                       ↪ .reset_index(drop=True)
                    df_sorted['Value'] = df_sorted['Value'].map('{:,.2f}'.
                       ↪ format)
```

Listing 2: The complete Python script used to generate the piece value lexicon.

C Comprehensive Piece Value Lexicon

Table 7: The complete list of calculated piece values, sorted by strength.

Short name	Name/Desciption	Value
BNR	Amazon	1,223.03
fnrama	Moves like Ferz+Knight+Rook, Captures like Amazon	1,200.77
chaama	Moves like Chancellor, Captures like Amazon	1,118.80
bnwama	Moves like Bishop+Knight+Wazir, Captures like Amazon	1,085.12
queama	Moves like Queen, Captures like Amazon	1,082.37
cenama	Moves like Centaur, Captures like Amazon	1,068.53
berama	Moves like Berserker, Captures like Amazon	1,062.80
arcama	Moves like Archbishop, Captures like Amazon	1,045.28
fnama	Moves like Ferz+Knight, Captures like Amazon	1,026.43
amafnr	Moves like Amazon, Captures like Ferz+Knight+Rook	1,011.89
FNR	Ferz+Knight+Rook	994.59
nwama	Moves like Knight+Wazir, Captures like Amazon	987.20
rooama	Moves like Rook, Captures like Amazon	982.72
draama	Moves like Dragon Horse, Captures like Amazon	952.12
amabnw	Moves like Amazon, Captures like Bishop+Knight+Wazir	948.33
kniama	Moves like Knight, Captures like Amazon	947.52
amaque	Moves like Amazon, Captures like Queen	942.59
manama	Moves like Man (King), Captures like Amazon	935.15
fnrbnw	Moves like Ferz+Knight+Rook, Captures like Bishop+Knight+Wazir	930.59
fnrque	Moves like Ferz+Knight+Rook, Captures like Queen	923.49
chafnr	Moves like Chancellor, Captures like Ferz+Knight+Rook	916.48
bisama	Moves like Bishop, Captures like Amazon	912.44
ferama	Moves like Ferz, Captures like Amazon	895.73
bnwfnr	Moves like Bishop+Knight+Wazir, Captures like Ferz+Knight+Rook	886.15
quefnr	Moves like Queen, Captures like Ferz+Knight+Rook	881.71
cenfnr	Moves like Centaur, Captures like Ferz+Knight+Rook	869.75
amacha	Moves like Amazon, Captures like Chancellor	865.81
berfnr	Moves like Berserker, Captures like Ferz+Knight+Rook	865.25
wazama	Moves like Wazir, Captures like Amazon	858.60
chabnw	Moves like Chancellor, Captures like Bishop+Knight+Wazir	854.27
fnrcha	Moves like Ferz+Knight+Rook, Captures like Chancellor	849.77
arcfnr	Moves like Archbishop, Captures like Ferz+Knight+Rook	847.53
chaque	Moves like Chancellor, Captures like Queen	847.23
fnfnr	Moves like Ferz+Knight, Captures like Ferz+Knight+Rook	831.20
BNW	Bishop+Knight+Wazir	824.90
quebnw	Moves like Queen, Captures like Bishop+Knight+Wazir	820.35
bnwque	Moves like Bishop+Knight+Wazir, Captures like Queen	817.84
BR	Queen	816.92
cenbnw	Moves like Centaur, Captures like Bishop+Knight+Wazir	808.67
berbnw	Moves like Berserker, Captures like Bishop+Knight+Wazir	804.22
cenque	Moves like Centaur, Captures like Queen	801.73
amaarc	Moves like Amazon, Captures like Archbishop	801.12
berque	Moves like Berserker, Captures like Queen	797.30
nwfnr	Moves like Knight+Wazir, Captures like Ferz+Knight+Rook	794.88
roofnr	Moves like Rook, Captures like Ferz+Knight+Rook	790.49
arcbnw	Moves like Archbishop, Captures like Bishop+Knight+Wazir	786.82
fnrarc	Moves like Ferz+Knight+Rook, Captures like Archbishop	785.49
arcque	Moves like Archbishop, Captures like Queen	779.93
NR	Chancellor	774.43
fnbnw	Moves like Ferz+Knight, Captures like Bishop+Knight+Wazir	770.88

Table 7 – continued from previous page

Betza	Name	Value
fnque	Moves like Ferz+Knight, Captures like Queen	764.06
drafnr	Moves like Dragon Horse, Captures like Ferz+Knight+Rook	761.69
amacen	Moves like Amazon, Captures like Centaur	757.99
knifnr	Moves like Knight, Captures like Ferz+Knight+Rook	757.29
amaber	Moves like Amazon, Captures like Berserker	751.22
bnwcha	Moves like Bishop+Knight+Wazir, Captures like Chancellor	746.79
manfnr	Moves like Man (King), Captures like Ferz+Knight+Rook	745.89
quecha	Moves like Queen, Captures like Chancellor	742.38
fnrcen	Moves like Ferz+Knight+Rook, Captures like Centaur	742.23
fnrber	Moves like Ferz+Knight+Rook, Captures like Berserker	735.53
nwbnw	Moves like Knight+Wazir, Captures like Bishop+Knight+Wazir	735.40
cencha	Moves like Centaur, Captures like Chancellor	730.98
roobnw	Moves like Rook, Captures like Bishop+Knight+Wazir	730.96
nwque	Moves like Knight+Wazir, Captures like Queen	728.58
bercha	Moves like Berserker, Captures like Chancellor	726.68
bisfnr	Moves like Bishop, Captures like Ferz+Knight+Rook	724.52
rooque	Moves like Rook, Captures like Queen	724.26
chaarc	Moves like Chancellor, Captures like Archbishop	712.79
arccha	Moves like Archbishop, Captures like Chancellor	709.78
ferfnr	Moves like Ferz, Captures like Ferz+Knight+Rook	708.91
drabnw	Moves like Dragon Horse, Captures like Bishop+Knight+Wazir	702.86
knibnw	Moves like Knight, Captures like Bishop+Knight+Wazir	698.57
draque	Moves like Dragon Horse, Captures like Queen	696.18
fncha	Moves like Ferz+Knight, Captures like Chancellor	694.27
amadra	Moves like Amazon, Captures like Dragon Horse	692.59
knique	Moves like Knight, Captures like Queen	691.88
manbnw	Moves like Man (King), Captures like Bishop+Knight+Wazir	687.38
bnwarc	Moves like Bishop+Knight+Wazir, Captures like Archbishop	684.84
manque	Moves like Man (King), Captures like Queen	680.70
quearc	Moves like Queen, Captures like Archbishop	680.55
fnrdra	Moves like Ferz+Knight+Rook, Captures like Dragon Horse	677.16
wazfnr	Moves like Wazir, Captures like Ferz+Knight+Rook	674.22
chacen	Moves like Chancellor, Captures like Centaur	670.73
cenarc	Moves like Centaur, Captures like Archbishop	669.41
bisbnw	Moves like Bishop, Captures like Bishop+Knight+Wazir	666.45
berarc	Moves like Berserker, Captures like Archbishop	665.17
chaber	Moves like Chancellor, Captures like Berserker	664.12
bisque	Moves like Bishop, Captures like Queen	659.88
nwcha	Moves like Knight+Wazir, Captures like Chancellor	659.78
roocha	Moves like Rook, Captures like Chancellor	655.57
ferbnw	Moves like Ferz, Captures like Bishop+Knight+Wazir	651.21
BN	Archbishop	654.40
ferque	Moves like Ferz, Captures like Queen	644.64
bnwcen	Moves like Bishop+Knight+Wazir, Captures like Centaur	643.20
quecen	Moves like Queen, Captures like Centaur	639.02
bnwber	Moves like Bishop+Knight+Wazir, Captures like Berserker	636.68
fnarc	Moves like Ferz+Knight, Captures like Archbishop	633.49
queber	Moves like Queen, Captures like Berserker	632.48
dracha	Moves like Dragon Horse, Captures like Chancellor	628.16
FNW	Centaur	628.79
amanw	Moves like Amazon, Captures like Knight+Wazir	624.72
knicha	Moves like Knight, Captures like Chancellor	624.00
bercen	Moves like Berserker, Captures like Centaur	623.88
cenber	Moves like Centaur, Captures like Berserker	621.56
amafn	Moves like Amazon, Captures like Ferz+Knight	621.39

Table 7 – continued from previous page

Betza	Name	Value
amaroo	Moves like Amazon, Captures like Rook	618.25
FR	Berserker	617.95
wazbnw	Moves like Wazir, Captures like Bishop+Knight+Wazir	617.29
mancha	Moves like Man (King), Captures like Chancellor	613.12
wazque	Moves like Wazir, Captures like Queen	610.79
fnrnw	Moves like Ferz+Knight+Rook, Captures like Knight+Wazir	609.73
arccen	Moves like Archbishop, Captures like Centaur	607.63
chadra	Moves like Chancellor, Captures like Dragon Horse	607.37
fnrfn	Moves like Ferz+Knight+Rook, Captures like Ferz+Knight	606.44
fnrroo	Moves like Ferz+Knight+Rook, Captures like Rook	603.28
arcber	Moves like Archbishop, Captures like Berserker	601.18
nwarc	Moves like Knight+Wazir, Captures like Archbishop	599.81
rooarc	Moves like Rook, Captures like Archbishop	595.69
bischa	Moves like Bishop, Captures like Chancellor	592.83
fncen	Moves like Ferz+Knight, Captures like Centaur	592.70
fnber	Moves like Ferz+Knight, Captures like Berserker	586.32
bnwdra	Moves like Bishop+Knight+Wazir, Captures like Dragon Horse	580.52
fercha	Moves like Ferz, Captures like Chancellor	577.99
quedra	Moves like Queen, Captures like Dragon Horse	576.43
draarc	Moves like Dragon Horse, Captures like Archbishop	568.93
cendra	Moves like Centaur, Captures like Dragon Horse	565.73
kniarc	Moves like Knight, Captures like Archbishop	564.85
berdra	Moves like Berserker, Captures like Dragon Horse	561.63
nwcen	Moves like Knight+Wazir, Captures like Centaur	559.53
amabis	Moves like Amazon, Captures like Bishop	559.22
roocen	Moves like Rook, Captures like Centaur	555.48
manarc	Moves like Man (King), Captures like Archbishop	554.22
nwber	Moves like Knight+Wazir, Captures like Berserker	553.22
roober	Moves like Rook, Captures like Berserker	549.18
arcdra	Moves like Archbishop, Captures like Dragon Horse	545.81
wazcha	Moves like Wazir, Captures like Chancellor	545.02
fnrbis	Moves like Ferz+Knight+Rook, Captures like Bishop	544.63
chanw	Moves like Chancellor, Captures like Knight+Wazir	541.69
chafn	Moves like Chancellor, Captures like Ferz+Knight	538.48
charoo	Moves like Chancellor, Captures like Rook	535.41
bisarc	Moves like Bishop, Captures like Archbishop	534.39
fndra	Moves like Ferz+Knight, Captures like Dragon Horse	531.21
dracen	Moves like Dragon Horse, Captures like Centaur	529.12
knicen	Moves like Knight, Captures like Centaur	525.12
draber	Moves like Dragon Horse, Captures like Berserker	522.88
ferarc	Moves like Ferz, Captures like Archbishop	519.89
amaman	Moves like Amazon, Captures like Man (King)	519.60
kniber	Moves like Knight, Captures like Berserker	518.89
bnwnw	Moves like Bishop+Knight+Wazir, Captures like Knight+Wazir	515.51
mancen	Moves like Man (King), Captures like Centaur	514.66
bnwfn	Moves like Bishop+Knight+Wazir, Captures like Ferz+Knight	512.31
quenw	Moves like Queen, Captures like Knight+Wazir	511.51
bnwroo	Moves like Bishop+Knight+Wazir, Captures like Rook	509.29
manber	Moves like Man (King), Captures like Berserker	508.46
quefn	Moves like Queen, Captures like Ferz+Knight	508.33
queroo	Moves like Queen, Captures like Rook	504.83
fnrman	Moves like Ferz+Knight+Rook, Captures like Man (King)	505.20
cennw	Moves like Centaur, Captures like Knight+Wazir	501.10
nwdra	Moves like Knight+Wazir, Captures like Dragon Horse	498.86
cenfn	Moves like Centaur, Captures like Ferz+Knight	497.94

Table 7 – continued from previous page

Betza	Name	Value
bernw	Moves like Berserker, Captures like Knight+Wazir	497.13
biscen	Moves like Bishop, Captures like Centaur	495.15
cenroo	Moves like Centaur, Captures like Rook	494.92
roodra	Moves like Rook, Captures like Dragon Horse	494.91
amakni	Moves like Amazon, Captures like Knight	494.77
berfn	Moves like Berserker, Captures like Ferz+Knight	493.97
berroo	Moves like Berserker, Captures like Rook	490.97
bisber	Moves like Bishop, Captures like Berserker	489.00
wazarc	Moves like Wazir, Captures like Archbishop	487.68
arcnw	Moves like Archbishop, Captures like Knight+Wazir	481.69
fercen	Moves like Ferz, Captures like Centaur	480.89
fnrkni	Moves like Ferz+Knight+Rook, Captures like Knight	480.52
arcfn	Moves like Archbishop, Captures like Ferz+Knight	478.55
chabis	Moves like Chancellor, Captures like Bishop	478.30
arcroo	Moves like Archbishop, Captures like Rook	475.56
ferber	Moves like Ferz, Captures like Berserker	474.75
BW	Dragon Horse	470.16
fnnw	Moves like Ferz+Knight, Captures like Knight+Wazir	467.48
knidra	Moves like Knight, Captures like Dragon Horse	465.28
FN	Ferz+Knight	464.73
fnroo	Moves like Ferz+Knight, Captures like Rook	461.40
mandra	Moves like Man (King), Captures like Dragon Horse	455.07
bnwbis	Moves like Bishop+Knight+Wazir, Captures like Bishop	452.77
wazcen	Moves like Wazir, Captures like Centaur	449.16
quebis	Moves like Queen, Captures like Bishop	452.00
wazber	Moves like Wazir, Captures like Berserker	443.13
chaman	Moves like Chancellor, Captures like Man (King)	439.94
cenbis	Moves like Centaur, Captures like Bishop	438.76
bisdra	Moves like Bishop, Captures like Dragon Horse	436.02
NW	Knight+Wazir	435.94
berbis	Moves like Berserker, Captures like Bishop	434.88
nwfn	Moves like Knight+Wazir, Captures like Ferz+Knight	432.87
roonw	Moves like Rook, Captures like Knight+Wazir	432.09
nwroo	Moves like Knight+Wazir, Captures like Rook	429.94
roofn	Moves like Rook, Captures like Ferz+Knight	429.02
R	Rook	425.66
ferdra	Moves like Ferz, Captures like Dragon Horse	422.10
arcbis	Moves like Archbishop, Captures like Bishop	419.81
chakni	Moves like Chancellor, Captures like Knight	415.91
bnwman	Moves like Bishop+Knight+Wazir, Captures like Man (King)	414.81
queman	Moves like Queen, Captures like Man (King)	410.06
dranw	Moves like Dragon Horse, Captures like Knight+Wazir	407.04
fnbis	Moves like Ferz+Knight, Captures like Bishop	405.98
drafn	Moves like Dragon Horse, Captures like Ferz+Knight	404.00
kninw	Moves like Knight, Captures like Knight+Wazir	403.23
draroo	Moves like Dragon Horse, Captures like Rook	401.11
cenman	Moves like Centaur, Captures like Man (King)	401.00
knifn	Moves like Knight, Captures like Ferz+Knight	400.19
amawaz	Moves like Amazon, Captures like Wazir	397.96
kniroo	Moves like Knight, Captures like Rook	397.31
berman	Moves like Berserker, Captures like Man (King)	397.21
amafer	Moves like Amazon, Captures like Ferz	394.94
mannw	Moves like Man (King), Captures like Knight+Wazir	393.28
wazdra	Moves like Wazir, Captures like Dragon Horse	391.15
bnwkni	Moves like Bishop+Knight+Wazir, Captures like Knight	391.03

Table 7 – continued from previous page

Betza	Name	Value
manfn	Moves like Man (King), Captures like Ferz+Knight	390.26
manroo	Moves like Man (King), Captures like Rook	387.39
quekni	Moves like Queen, Captures like Knight	386.80
fnrwaz	Moves like Ferz+Knight+Rook, Captures like Wazir	384.28
arcman	Moves like Archbishop, Captures like Man (King)	382.38
fnrfer	Moves like Ferz+Knight+Rook, Captures like Ferz	381.26
cenkni	Moves like Centaur, Captures like Knight	377.38
nwbis	Moves like Knight+Wazir, Captures like Bishop	375.25
bisnw	Moves like Bishop, Captures like Knight+Wazir	374.72
berkni	Moves like Berserker, Captures like Knight	373.61
bisfn	Moves like Bishop, Captures like Ferz+Knight	371.72
roobis	Moves like Rook, Captures like Bishop	371.49
bisroo	Moves like Bishop, Captures like Rook	368.87
fnman	Moves like Ferz+Knight, Captures like Man (King)	368.76
fernw	Moves like Ferz, Captures like Knight+Wazir	361.15
arckni	Moves like Archbishop, Captures like Knight	358.93
ferfn	Moves like Ferz, Captures like Ferz+Knight	358.17
ferroo	Moves like Ferz, Captures like Rook	355.34
drabis	Moves like Dragon Horse, Captures like Bishop	347.08
fnkni	Moves like Ferz+Knight, Captures like Knight	345.45
knibis	Moves like Knight, Captures like Bishop	343.36
nwman	Moves like Knight+Wazir, Captures like Man (King)	338.51
rooman	Moves like Rook, Captures like Man (King)	334.82
manbis	Moves like Man (King), Captures like Bishop	333.67
waznw	Moves like Wazir, Captures like Knight+Wazir	331.00
wazfn	Moves like Wazir, Captures like Ferz+Knight	328.06
wazroo	Moves like Wazir, Captures like Rook	325.26
chawaz	Moves like Chancellor, Captures like Wazir	322.23
chafer	Moves like Chancellor, Captures like Ferz	319.30
B	Bishop	317.58
nwkni	Moves like Knight+Wazir, Captures like Knight	315.51
rookni	Moves like Rook, Captures like Knight	311.42
draman	Moves like Dragon Horse, Captures like Man (King)	310.79
kniman	Moves like Knight, Captures like Man (King)	307.13
ferbis	Moves like Ferz, Captures like Bishop	302.36
bnwwaz	Moves like Bishop+Knight+Wazir, Captures like Wazir	298.33
FW	Man (King)	298.54
bnwfer	Moves like Bishop+Knight+Wazir, Captures like Ferz	295.43
quewaz	Moves like Queen, Captures like Wazir	294.69
quefer	Moves like Queen, Captures like Ferz	291.80
drakni	Moves like Dragon Horse, Captures like Knight	288.06
cenwaz	Moves like Centaur, Captures like Wazir	285.21
N	Knight	286.19
cenfer	Moves like Centaur, Captures like Ferz	282.33
berwaz	Moves like Berserker, Captures like Wazir	281.60
bisman	Moves like Bishop, Captures like Man (King)	279.79
berfer	Moves like Berserker, Captures like Ferz	278.72
mankni	Moves like Man (King), Captures like Knight	275.00
wazbis	Moves like Wazir, Captures like Bishop	272.97
arcwaz	Moves like Archbishop, Captures like Wazir	267.50
ferman	Moves like Ferz, Captures like Man (King)	266.77
arcfer	Moves like Archbishop, Captures like Ferz	263.64
biskni	Moves like Bishop, Captures like Knight	259.33
fnwaz	Moves like Ferz+Knight, Captures like Wazir	254.56
fnfer	Moves like Ferz+Knight, Captures like Ferz	251.72

Table 7 – continued from previous page

Betza	**Name**	**Value**
ferkni	Moves like Ferz, Captures like Knight	244.49
wazman	Moves like Wazir, Captures like Man (King)	237.85
nwwaz	Moves like Knight+Wazir, Captures like Wazir	225.80
nwfer	Moves like Knight+Wazir, Captures like Ferz	222.99
roowaz	Moves like Rook, Captures like Wazir	222.29
roofer	Moves like Rook, Captures like Ferz	219.49
wazkni	Moves like Wazir, Captures like Knight	215.86
drawaz	Moves like Dragon Horse, Captures like Wazir	199.44
drafer	Moves like Dragon Horse, Captures like Ferz	196.67
kniwaz	Moves like Knight, Captures like Wazir	195.96
knifer	Moves like Knight, Captures like Ferz	193.19
manwaz	Moves like Man (King), Captures like Wazir	186.89
manfer	Moves like Man (King), Captures like Ferz	184.14
biswaz	Moves like Bishop, Captures like Wazir	169.97
bisfer	Moves like Bishop, Captures like Ferz	167.23
ferwaz	Moves like Ferz, Captures like Wazir	157.59
F	Ferz	156.91
W	Wazir	132.25
wazfer	Moves like Wazir, Captures like Ferz	129.58