Minimizing Inconsistency in Pairwise Comparison Matrices Using Genetic Algorithm

Atiyeh Sayadi
McMaster University
Hamilton, Ontario, Canada
sayadia@mcmaster.ca

Baran Shajari
McMaster University
McMaster University, Canada
McMasterUniversity

Abstract

This paper introduces a novel approach to reduce inconsistencies in pairwise comparison matrices using a genetic algorithm inspired by the process of natural selection. The method applies a distance-based inconsistency index as the fitness function within an evolutionary process. Through experiments, we show that our genetic algorithm greatly reduces inconsistencies over generations. Smaller matrices quickly become consistent in just a few generations, showing the method's effectiveness and efficiency for different scenarios and use cases. Larger matrices, however, require more generations to reach the acceptable consistency level. The algorithm works well for different matrix sizes, adapting effectively to various challenging situations.

CCS Concepts

• **Evolutionary Algorithm**; • **Genetic Algorithm**; • **Pairwise Comparison**;

Keywords

Pairwise Comparison, Evolutionary Algorithm, Inconsistency Reduction, Distance-Based Inconsistency, Genetic Algorithm.

ACM Reference Format:
Atiyeh Sayadi and Baran Shajari. 2024. Minimizing Inconsistency in Pairwise Comparison Matrices Using Genetic Algorithm. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The pairwise comparison method is based on the observation that it is much easier to rank the importance of two objects than it is to rank the importance of several objects [7, 19]. Pairwise comparison is frequently used for decision making, especially when subjective judgment is involved. This method simplifies complex decisions by breaking them down into binary comparisons, making it highly valuable in decision-making problems.

Pairwise comparison was used even before the invention of numbers, dating back to the Stone Age, when people needed to weigh two goods for exchange. For example, they might have held a fish in one hand and a bird in the other to compare their relative weights. It is one of the oldest scientific methods that traced its way back to 1274–1283 works of the medieval philosopher Ramon Llull. The method was later independently reinvented by de Condorcet in 1785. In both cases, it was used as a voting method [11, 15, 19, 23].

It is commonly used in ranking problems, where a ranking system is built based on the results of each pairwise comparison. In other words, a ranking system is created from the combined results of comparing all pairs of items, often represented as a matrix. However, it is important to consider inconsistencies in pairwise comparisons, as they can impact the accuracy and reliability of the results.

For example, one might get $A \prec B, B \prec C$ and $C \prec A$ ('$A \prec B$' means B is preferred over A), or even $A \prec B$ and $B \prec A$ (cf. [17]). There are two popular *inconsistency indexes*, one proposed by Saaty in 1977 [23] (the value of the largest eigenvalue of pairwise comparisons matrix), and another, proposed by Koczkodaj [15] (based on the analysis of triads). Their differences and similarities have been discussed in detail in [3]. The basic advantage of Koczkodaj's index is that the algorithms for minimizing inconsistency are simpler, more intuitive, and more efficient [3, 16–18]. This index will be used in this paper.

A genetic algorithm is a method inspired by nature, used to find a global optimum solution by mimicking the process of natural selection [5]. Genetic algorithms are usually divided into two types: single-solution metaheuristic algorithms, which improve one solution step by step, and population-based metaheuristic algorithms, which work with many solutions at the same time. The algorithm in this paper is population-based because it works with a group of solutions that evolve over several generations.

In population-based genetic algorithms, a fitness function is used to measure how good each solution in the population is. This function helps decide which solutions to keep and which to discard, based on the idea of "survival of the fittest" [5]. The goal of the fitness function depends on the problem, and it is used to minimize or maximize a specific objective.

In this paper, we use a genetic algorithm to reduce the inconsistency of pairwise comparison matrices. The process starts with one initial matrix, and the first population is created by mutating this matrix. Over the generations, the solutions are improved using crossover and mutation. For selection, we use an elitism strategy to make sure the best solutions are carried to the next generation. The goal is to decrease the inconsistency metric of the matrix, making it more consistent. By following this evolutionary process, the algorithm reduces inconsistency effectively while keeping the method computationally efficient.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

This paper is structured as follows. We first present related concepts and definitions of pairwise comparison, including inconsistency, along with a brief overview of genetic algorithms. We then introduce a genetic algorithm for inconsistency reduction, providing all results accompanied by charts. Furthermore, we compare our work to other similar studies addressing this problem.

2 Pairwise Comparisons and Consistency

In general, there are two kinds of pairwise comparisons, *quantitative* and *qualitative*. Quantitative pairwise comparisons use numbers to describe the preference relationship between objects [4, 7, 15, 23], while qualitative pairwise comparisons use abstract relations (as 'indifferent', 'slightly in favour', 'better' etc.) for description of this relationship [10, 14]. Quantitative pairwise comparisons can either be *additive* [4, 13], or, more popular *multiplicative* [19, 23]. All three kinds of pairwise comparisons are compared and analyzed in [11]. In this paper, we will deal only with *quantitative multiplicative* pairwise comparisons since they have the most advanced methods for analysis and enforcement of consistency. The other models of pairwise comparisons are usually transformed into quantitative multiplicative ones when finding acceptable inconsistency is an issue to deal with [11, 13].

Let $E_1, ..., E_n$ be a finite set of objects (entities) to be judged and/or analyzed. The quantitative relationship between entities E_i and E_j is represented by a positive number a_{ij}. We assume $a_{ij} > 0$ and $a_{ij} = \frac{1}{a_{ji}}$, for $i, j = 1, ..., n$ (which implies $a_{ii} = 1$ for all i). If $a_{ij} > 1$ then E_i is more important (preferred, better, etc.) than E_j and a_{ij} is a measure of this relationship (the bigger a_{ij}, the bigger the difference), if $a_{ij} = 1$ then E_i and E_j are indifferent. We call this model *multiplicative* since a_{ij} is interpreted as E_j *is* a_{ij} *times preferred* (*more important, etc.*) *than* E_j.

The matrix of such (multiplicative) relative comparison coefficients,

$$A = \begin{bmatrix} 1 & a_{12} & \dots & a_{1n} \\ \frac{1}{a_{12}} & 1 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{a_{1n}} & \frac{1}{a_{2n}} & \dots & 1 \end{bmatrix}. \tag{1}$$

is called a (multiplicative) *pairwise comparison matrix* [23].

A pairwise comparison matrix $A = [a_{ij}]_{n\times n}$ is *consistent* [23] if and only if

$$a_{ij}.a_{jk} = a_{ik} \tag{2}$$

for $i, j, k = 1, ..., n$.

Saaty's Theorem [23] states that a pairwise comparison matrix A is consistent if and only if there exist positive numbers $w_1, ..., w_n$ such that $a_{ij} = w_i/w_j$, $i, j = 1, ..., n$. The values w_i are unique up to a multiplicative constant. They are often called *weights* and interpreted as a measure of importance. Weights may be scaled to $w_1+..+w_n = 1$ (or 100%) and they obviously create 'natural' ranking ($E_i < E_j \iff w_i < w_j$ and $E_i \approx E_j \iff w_i = w_j$). In practice, the values a_{ij} are very seldom consistent, so some measurements of inconsistency, and a guidance what level of inconsistency is accepted, are needed.

The first step in this procedure is to assign numerical value a_{ij} that measures the relationship between E_i and E_j on the basis of some *subjective* judgment. For this, a proper *scale* is needed. The numbers from a given scale are used as initial values of a_{ij}'s. A linear ten points scale '$1, \dots, 10$' (or the interval $[\frac{1}{10}, 10]$ for almost consistent final a_{ij}'s) was proposed in [23]. It is still often used. Unfortunately, it is not trustworthy since it does not take the limitations of the human mind into consideration. According to [6, 22], the length of the scale should be between four [6] and seven alternatives [22]. Five-point scales have been proposed and advocated in [3, 11, 13, 15]. In practice, the scale is domain-oriented; different applications require different scales. In this paper, the seven-point scale suggested in [22], is used.

Saaty [23] proposed an inconsistency index based on the value of the largest eigenvalue of A. The basic problem is that *it does not give any clue where most inconsistent values of A are located* [3, 13, 15]. On the other hand, *distance-based inconsistency* [15], for a given $A = [a_{ij}]_{n\times n}$, defined as:

$$cm_A = \max_{(i,j,k)} \left(\min \left(\left| 1 - \frac{a_{ij}}{a_{ik}.a_{kj}} \right|, \left| 1 - \frac{a_{ik}.a_{kj}}{a_{ij}} \right| \right) \right) \tag{3}$$

localizes the most inconsistent triad, so we can reduce inconsistency by some minor changes a_{ij}, a_{ik}, a_{kj}. It is often assumed that the matrix is sufficiently consistent when the inconsistency value is at most 0.3 [15]. We will use this value in our algorithm.

A fast algorithm for inconsistency reduction has been proposed [16]. Recently this algorithm has been used in the online tool PiXR [12]. However, PiXR tool has some limitations, such as its algorithms struggling to quickly identify extreme inconsistencies as the number of parameters increases [12].

3 Genetic Algorithm

A genetic algorithm is a subfield of evolutionary computation, inspired by nature and biological evolution, to solve complex computational problems [9]. They refine solutions by mimicking biological mechanisms of evolution [24]. This approach is based on trial-and-error, much like the natural process of evolution. In this method, a population of individuals represents potential solutions that evolve over iterations through processes similar to natural selection, mutation, and crossover. The evolutionary algorithm incorporates concepts of genotype and phenotype based on its biological foundation: the genotype (internal encoding) determines the phenotype (external characteristics). Genes act as the functional units of inheritance, encoding phenotypic traits [9]. In genetic algorithms, potential solutions (individuals) are represented as "genetic" encoding that undergo mutation and crossover (recombination) to generate the next generation. Crossover combines genes from two parents to produce offspring, while mutation introduces small changes to individuals. A fitness function evaluates each individual, determining whether a solution is "fit" enough to survive to the next generation or should be removed. This approach enables genetic algorithms to solve a range of problems, from optimization to machine learning, by adapting and refining solutions in ways similar to the diversity and specialization observed in biological species.

Further expanding on genetic algorithms, they are domain independent and robust methods which makes them ideal for large, complex and poorly understood search spaces [8]. Genetic algorithms have evolutionary mechanisms that allow them to emerge through mutations, crossovers, and selections to achieve a solution

[1]. They are effective optimization methods that have succeeded in complex problems and are recognized as promising tools for problem-solving [1]. By using an iterative trial-and-error approach, they increase the adaptability of genetic algorithms in dynamic problem spaces [1, 24]. Genetic algorithms ability to attain information about an unknown search space makes them highly valuable for problems with limited domain specific knowledge [8].

Some examples of success shown by genetic algorithms in dynamic problem spaces were in the areas of operation management and production [1]. Genetic algorithms leverage their genotype-phenotype relationship to optimize workflow, which is applied to different complex scheduling tasks [1]. Genetic algorithms are also used in other fields, such as medicine, to minimize differences and align images in applications like X-rays or optimize link speed in a particular topology to design digital communication networks [20]. These examples show the high adaptability of genetic algorithms for high dimensional problems [20].

4 Genetic Algorithm for Consistency

The starting point of our algorithm is a given pairwise comparison matrix $A = [a_{ij}]_{n\times n}$ filled with initial values of a_{ij} based on provided subjective judgments.

4.1 Search and Selection

In this research, matrices are leveraged to manipulate and store data which allow us to analyze data efficiently. Matrices provide us with the flexibility to work with various data types and is efficient for diverse applications. In addition, the initial population is not derived entirely from random numbers but is gained from our own carefully curated entries which ensures that it has contextual relevance and can be used in real-life scenarios. In our algorithm, we carefully used crossover and mutation techniques to ensure the matrix does not undergo a lot of changes. These operations are precise to maintain the originality of our data.

In our algorithm, different rates for mutation and crossover as well as population size was tested until an optimal algorithm was reached that yields the best results. Moreover, these data have undergone mutations themselves, and the iterative crossovers and mutations applied to our data have allowed us to refine our algorithm consistently.

We have evaluated the matrices based on an inconsistency value of 0.3, as suggested in [15] and often used in practice [11]. If the inconsistency is below 0.3 in value, the matrix is considered consistent and if the consistency level is over 0.3, it indicates that the result is not consistent and cannot be trusted as a reliable source for decision making. Moreover, this threshold act as a benchmark for evaluating inconsistency of different size matrices.

To evaluate this relationship, 100 matrices were generated for each size to assess the impact of the matrices size on the number of generations needed to reach the targeted inconsistency.

4.2 Algorithm Overview

We start with an input matrix A with a size of at least 3×3, where based on the classical result [22], we will use the continuous scale $[\frac{1}{7}, 7]$. From this input, we generate 999 offspring by mutating a single randomly selected element in the matrix. The mutation process adheres to the following rules:

- The indices i and j must satisfy $i \neq j$.
- The mutated value a_{ij} is updated, ensuring the relationship $a_{ij}.\ a_{ji} = 1$ holds after mutation. Therefore, a_{ji} also needs to be modified accordingly.
- The mutation involves adding a random value, x, chosen from the interval $[-0.5, 0.5]$, to a_{ij}. After adding x, the result is checked to ensure it stays within the valid range $\left[\frac{1}{7}, 7\right]$. The range $[-0.5, 0.5]$ is carefully chosen because each qualitative relationship, such as equality or preference, corresponds to a specific quantitative range. Indifference ($a \approx b$), for example, has a narrow range of $[0.79, 1.27]$ (based on [11, 22]), meaning even small changes can shift a relationship from $a \sqsubset b$ (slightly in favor of b) to $b \sqsubset a$ (slightly in favor of a), skipping the indifference. A larger range for x could cause such shifts, disrupting the intended relationships, while a smaller range would limit the algorithm's ability to explore new solutions efficiently. Thus, $[-0.5, 0.5]$ ensures meaningful but controlled variations that preserve qualitative relationships while allowing optimization.

This gives us an initial population of 1000 matrices. From here, we iterate through the evolutionary process until one of the following conditions is met:

- The inconsistency measure of the best matrix is less than 0.3.
- The number of generations exceeds 100.

The steps of the evolutionary loop are as follows:

(1) **Evaluation:** Each matrix in the population is evaluated using a fitness function, which measures inconsistency.
(2) **Selection:** The top 50% of matrices with the lowest inconsistency are selected.
(3) **Elitism:** 20% of the top matrices are randomly chosen and passed directly to the next generation to preserve high-quality solutions.
(4) **Crossover:** The remaining 90% of the next generation are produced by performing crossover on selected parent matrices.
(5) **Direct Transfer:** Additionally, 10% of the offspring are passed directly to the next generation to maintain valuable characteristics.
(6) **Mutation:** The rest of the population is filled by mutating matrices after crossover, where only one randomly selected element is changed by adding a number from the interval $[-0.5, 0.5]$.

Consider the following example from [2]:

$$A = \begin{bmatrix} 1 & 2 & 5 & 9 \\ 0.5 & 1 & 3 & 8 \\ 0.2 & 1/3 & 1 & 4 \\ 1/9 & 0.125 & 0.25 & 1 \end{bmatrix}$$

which has the inconsistency index $cm_A = 0.55 > 0.3$, so A is inconsistent. Our algorithm, after two generations, produces the

matrix

$$A_{0.3} = \begin{bmatrix} 1 & 1.2964 & 5 & 9 \\ 0.7714 & 1 & 3 & 8 \\ 0.2 & 1/3 & 1 & 2.0488 \\ 1/9 & 0.125 & 0.4881 & 1 \end{bmatrix}$$

which has the inconsistency index $cm_{A_{0.3}} = 0.2317 < 0.3$, i.e. $A_{0.3}$ is considered to be consistent.

4.3 Test Results

"Fig.1" is the resulting chart from a 7×7 matrix, which shows how the fitness value decreases as the number of generations increases, indicating that the fitness value is improving. Hence, it shows how the data becomes optimized and more consistent as the number of generations increases. In addition, the classic optimization curve is shown by this graph, where initial generations improve faster. However, as the number of generations increases, the improvements become marginal as the graph approaches the near-optimal or optimal stage.

A 3×3 matrix was tested and shown that it only takes 1-2 generations for it to reach consistency. In addition, "Fig. 2" is the resulting chart from a 4×4 matrix. The x-axis shows the number of generations, while the y-axis represents the number of matrices required to achieve consistency in various generation counts. The chart shows how consistency is achieved in the matrix with about 2-7 generations, with 3 generations being the peak count where the values stabilize. There is a small number of matrices requiring only 1 generation or more than 7 generations to reach consistency. This suggests the process is usually efficient since most matrices stabilize in a moderate number of generations. However, as the matrix size was increased in "Fig. 3", there were fewer matrices that reached consistency in early generations (33 to 40). The majority of matrices require a moderate number of generations (40 to 60) to reach consistency, with peaks observed around 49-52 and 56-57 generations. Some matrices take much longer to achieve consistency, with relatively fewer matrices reaching it in the range of 65 to 70 generations.

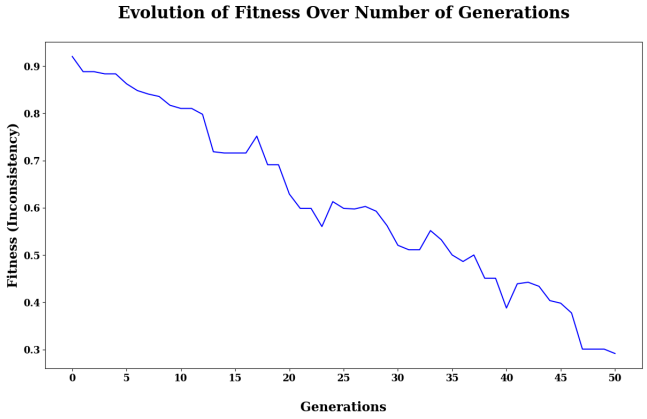

Figure 1: *Graph showing the evolution of fitness over a number of generations in a 7×7 matrix.*

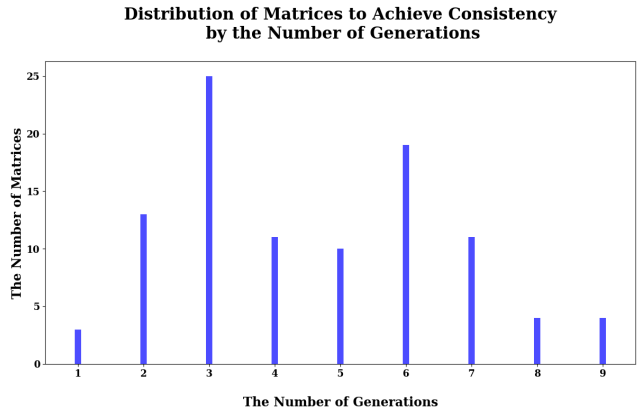

Figure 2: *Graph showing the distribution of matrices to achieve consistency by the number of generations in a 4×4 matrix.*

4.4 Results Analysis

The evolutionary process results illustrate that the matrix inconsistency decreases consistently as the number of generations increases. However, larger matrices require more generations in order to decrease their inconsistency to a value below 0.3. For example, suppose a 4×4 matrix is used. In that case, the evolutionary process gets completed within 10 generations, but 5×5, 6×6 and 7×7 matrices take more generations progressively to complete this process since they are larger. The method used in this algorithm reinforces the idea that it is often not necessary to work with large matrices since pairwise comparison is mainly based on human judgment who are not able to process large matrices. This further highlights how larger matrices will have increased complexity. When comparing our approach to the previous approaches, which produced their random set from fully consistent matrices [16], our initial data is captured directly from an expert which is not entirely random and contains a large percentage of inconsistencies, which are only slightly mutated. This approach allowed us to maintain some of the original structure from the expert input. However, the chances of generating at least one matrix with nearly acceptable consistency increase if we were to produce the first population (1000 matrices) from a fully consistent matrix.

5 Conclusion

The previous approach proposed in [16] and also used in [11, 12], reached consistency usually within 10 iterations or less, however this was due to choosing a fully consistent matrix as an initial dataset which means the matrix was close to consistency from the beginning. However, we have used a dataset provided from an expert which meant that we did not use a fully random initial matrix, but it was ensured that the initial dataset has a large percentage of inconsistencies to start with. This approach makes our finding more reliable and applicable by allowing us to align the dataset with real-life scenarios.

From our analysis, we can conclude that the size of the population greatly impacts the algorithm's performance. The result illustrated that as the population size increases, we see a decrease in the number of generations needed to reach the target inconsistency,

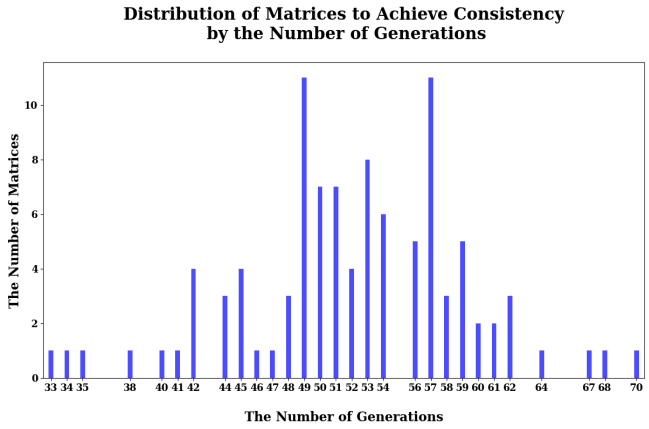

Figure 3: *Graph showing the distribution of matrices to achieve consistency by the number of generations in a* 7×7 *matrix.*

meaning that the larger the population, the quicker the algorithm convergence.

In addition, if the number of matrices passed directly from the crossover (with no mutation) and elites get reduced, the convergence of the algorithm to reach the target inconsistency will accelerate. However, maintaining parts of the original input matrix data is crucial to retaining valuable information. It is essential to mention that mutation also plays a vital role in improving the result; if we increase the mutation range above 0.5, improved performance is observed, but it is usually not necessary to go under excessive change.

This paper is our first experience with applying genetic algorithms to pairwise comparison paradigm and we plan to continue this research direction. Our first, short term goal, is to investigate the relationship between the inconsistency threshold and the number of generations. We assume the inconsistency threshold as 0.3, which is used very often, however for some specific applications lower values as 0.2 or even 0.1 might be more convincing [11]. Our second, also short term goal is to involve genetic algorithms for pairwise comparisons with incomplete information (as in [21]). A more challenging long term goal is to use genetic algorithms to manage directly inconsistency for qualitative models as that of [10, 14], which eventually may result is proposing better scale that all currently used.

References

[1] Haldun Aytug, Moutaz Khouja, and F. Elizabeth Vergara. 2003. Use of genetic algorithms to solve production and operations management problems: A review. *International Journal of Production Research* 41 (2003), 3955–4009. https://api.semanticscholar.org/CorpusID:16023946

[2] Sandor Bozoki. 2008. Solution of the least squares method problem of pairwise comparison matrices. *CEJOR* 16 (2008), 348–358.

[3] Sándor Bozóki and Tamás Rapcsák. 2008. On Saaty's and Koczkodaj's inconsistencies of pairwise comparison matrices. *Journal of Global Optimization* 42 (10 2008), 157–175. https://doi.org/10.1007/s10898-007-9236-z

[4] Ralph A. Bradley and Terry Milton E. 1952. The rank analysis of incomplete block designs. I. The method of paired comparisons. *Biometrika* 39 (1952), 324–345.

[5] Naci Caglar, Aydin Demir, Hakan Ozturk, and Abdulhalim Akkaya. 2015. A simple formulation for effective flexural stiffness of circular reinforced concrete columns. *Engineering Applications of Artificial Intelligence* 38 (2015), 79–87. https://doi.org/10.1016/j.engappai.2014.10.011

[6] Nelson Cowan. 2001. The magical number four in short-term memory. A reconsideration of mental storage capacity. *Behavioural and Brain Sciences* 24 (2001), 87–185.

[7] Herbert A. David. 1988. *The Method of Paired Comparisons.* Oxford University Press.

[8] Kenneth De Jong. 1988. Learning with Genetic Algorithms: An Overview. *Machine Learning* 3 (10 1988), 121–138. https://doi.org/10.1007/BF00113894

[9] A. E. Eiben and J. Smith. 2015. *Introduction to evolutionary computing.* Natural Computing Series.

[10] Ryszard Janicki. 2009. Pairwise Comparisons Based Non-Numerical Ranking. *Fundam. Inform.* 94 (01 2009), 197–217. https://doi.org/10.3233/FI-2009-126

[11] Ryszard Janicki. 2018. Finding consistent weights assignment with combined pairwise comparisons. *International Journal of Management and Decision Making* 17, 3 (2018), 322–347. https://ideas.repec.org/a/ids/ijmdma/v17y2018i3p322-347.html

[12] Ryszard Janicki and Mahmoud Mahmoud. 2022. On Multiplicative, Additive and Qualitative Pairwise Comparisons. 247–251. https://doi.org/10.15439/2022F57

[13] Ryszard Janicki and Yun Zhai. 2011. Remarks on Pairwise Comparison Numerical and Non-numerical Rankings. In *Rough Sets and Knowledge Technology*, JingTao Yao, Sheela Ramanna, Guoyin Wang, and Zbigniew Suraj (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 290–300.

[14] Ryszard Janicki and Yun Zhai. 2012. On a pairwise comparison-based consistent non-numerical ranking. *Logic Journal of the IGPL* 4 (08 2012). https://doi.org/10.1093/jigpal/jzr018

[15] Waldemar W. Koczkodaj. 1993. A new definition of consistency of pairwise comparisons. *Mathematical and Computer Modelling* 18, 7 (1993), 79–84. https://doi.org/10.1016/0895-7177(93)90059-8

[16] Waldemar W. Koczkodaj, Marek Kosiek, Jacek Szybowski, and Ding Xu. 2015. Fast Convergence of Distance-based Inconsistency in Pairwise Comparisons. *Fundam. Informaticae* 137 (2015), 355–367. https://api.semanticscholar.org/CorpusID:2047999

[17] Waldemar W. Koczkodaj and Marian Orłowski. 1999. Computing a Consistent Approximation to a Generalized Pairwise Comparisons Matrix. *An International Journal of Computers and Mathematics with Applications* 37 (1999), 79–85.

[18] Konrad Kułakowski. 2015. Notes on order preservation and consistency in AHP. *European Journal of Operational Research* 245, 1 (2015), 333–337.

[19] Konrad Kułakowski. 2020. *Understanding the analytic hierarchy process.* CRC Press.

[20] G. E. Liepins and M. R. Hilliard. 1989. Genetic algorithms: Foundations and applications. *Annals of Operations Research* 21, 1 (1989), 31–57.

[21] Jiri Mazurek and Ryszard Janicki. 2024. A Hybrid Stochastic-Full Enumeration Approach to a Ranking Problem with Insufficient Information. In *Proc. of 58th Annual Conference on Information Sciences and Systems (CISS), Princeton, NJ, USA.* 1–6.

[22] George A. Miller. 1956. The magical number seven, plus or minus two. *The Psychological Review* 63, 2 (1956), 81–97.

[23] Thomas L Saaty. 1977. A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology* 15, 3 (1977), 234–281. https://doi.org/10.1016/0022-2496(77)90033-5

[24] Luca Scrucca. 2013. GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software* 53, 4 (2013), 1–37. https://doi.org/10.18637/jss.v053.i04