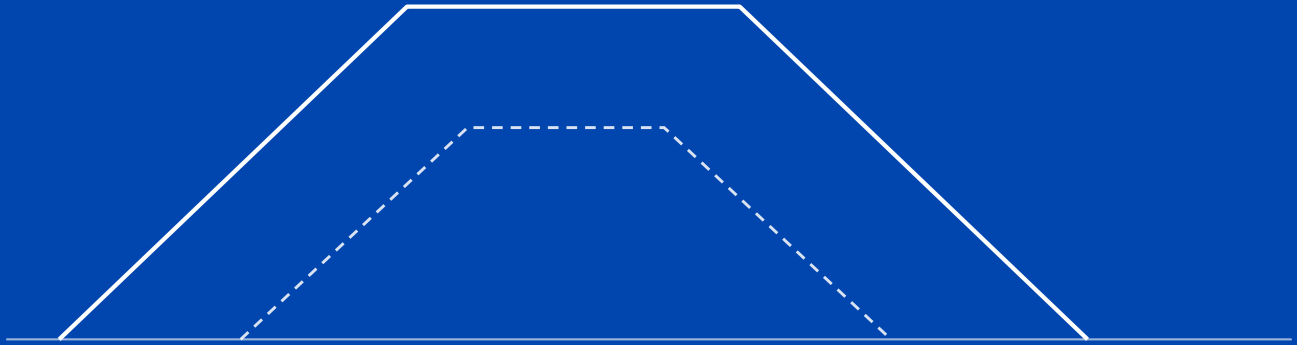




Fuzzy MCDM

Technical and Scientific Documentation

Criteria weighting with crisp, fuzzy and interval type-2 fuzzy AHP

Version 1.0.0

2026-10-15

Ing. Akram Elsaied Elomiya

ORCID <https://orcid.org/0000-0002-7166-0239> University of Pardubice, Faculty of Transport Engineering, Department of Transport Management, Marketing, and Logistics, Studentská 95, 532 10 Pardubice, Czech Republic

Tanta University, Faculty of Engineering, Department of Public Works Engineering, 31511, Tanta, Egypt

Supervisor: doc. Ing. Jiří Křupka, Ph.D.

This software is supported by the project SGS_2026_008 'Modelling of Transport Technology and Management Selected Aspects VI' at the University of Pardubice.

Fuzzy MCDM 1.0.0

© 2026 University of Pardubice. Distributed under the Fuzzy MCDM Research Licence 1.0.

Website: <https://fuzzy-mcdm.pages.dev>

DOI: 10.5281/zenodo.23380643

This software is supported by the project SGS_2026_008 'Modelling of Transport Technology and Management Selected Aspects VI' at the University of Pardubice.

Microsoft and Windows are trademarks of the Microsoft group of companies. Other names may be trademarks of their respective owners.

Abstract

Fuzzy MCDM is an offline desktop application for Windows that derives criteria weights for multi-criteria decision making from pairwise expert judgments. From one shared set of judgments it computes weights with five methods: the analytic hierarchy process with the eigenvector priority of Saaty, fuzzy AHP after Buckley (1985), fuzzy AHP by the fuzzy inverse of column sum method of Ahmed and Kilic (2019), interval type-2 fuzzy AHP after Kahraman et al. (2014) with the full 1 to 9 scale of Beresneva and Gordenko (2018), and interval type-2 fuzzy AHP with the scale of Milošević et al. (2019). One consistency gate applies to every method, so the weight vectors differ only because of the methods. The software compares the methods, measures expert disagreement, recomputes all weights without each expert in turn and estimates their stability with a seeded bootstrap over experts. Every number can be traced to the raw judgments and every equation, scale and constant to the page, table or equation of the publication that defines it; every run carries a hash-based identity.

This document describes the design of the software, the input model, the consistency policy, each method with its equations and sources, the analyses, the traceability and reproducibility mechanisms, and the verification against published worked examples. It closes with the novelty of the software with respect to existing tools and with its limitations. It is the technical documentation of the result “Fuzzy MCDM” of type R (software).

Conventions

Interface terms appear as **Page or button**, identifiers of methods, profiles and rules as IDENTIFIER, and files as name.ext. A comparison set is a group of criteria that the experts compare with each other: the main criteria, or the subcriteria of one main criterion. Its size is n . Experts are indexed by e , criteria by i and j . Equations are numbered by chapter.

Contents

Abstract	iii
1 Introduction	1
1.1 Criteria weighting and the choice of method	1
1.2 What the software does	1
1.3 Scope of this document	1
2 Design principles	3
2.1 One input, many methods	3
2.2 The same rules for every method	3
2.3 Fail closed	3
2.4 Every number has a source	3
2.5 Frozen, versioned science	3
2.6 Offline and private by construction	4
3 Architecture	5
3.1 Runtime	5
3.2 Offline design and its enforcement	5
3.3 Package structure	6
3.4 Data flow of a run	8
3.5 The study container	9
3.6 Persistence and recovery	10
3.7 Method profiles and the source registry	10
4 Input model	13
4.1 The study hierarchy	13
4.2 Experts and influence weights	13
4.3 Pairwise judgments	14
4.4 The signed text grammar of the import	15
4.5 Worked example: one expert's judgments	15
4.6 Eligibility of the computed weights	16
5 Consistency policy	17
5.1 The principal eigenvalue	17
5.2 Consistency index and consistency ratio	17
5.3 The random index table	18
5.4 The acceptance threshold	18
5.5 The gate and its rules	18
5.6 Group consistency ratio	19
5.7 Worked example: consistency of one expert	19
6 Crisp and triangular fuzzy AHP methods	21
6.1 Overview	21
6.2 The shared geometric group step	21
6.3 AHP, the eigenvector method	22
6.4 The row geometric mean internal check	23

6.5	The triangular fuzzy mapping	23
6.6	Fuzzy AHP, the Buckley geometric mean	23
6.7	Fuzzy AHP, the FICSM robustness variant	25
6.8	The four methods on one example	25
7	Interval type-2 fuzzy AHP methods	27
7.1	Overview	27
7.2	The interval type-2 number and the shared operators	27
7.2.1	Reciprocal	28
7.2.2	Group aggregation	28
7.2.3	Row geometric mean	28
7.2.4	Sum and division	28
7.2.5	Defuzzification and normalisation	29
7.2.6	Group consistency ratio	29
7.3	IT2 fuzzy AHP, the full 1–9 composition	30
7.3.1	The scale	30
7.3.2	Limitations	31
7.3.3	Golden cases	31
7.4	IT2 fuzzy AHP, the historical frozen reference	31
7.4.1	The scale	31
7.4.2	Limitations	31
7.4.3	Golden cases	32
7.5	Check against the published example	32
8	Synthesis and analyses	35
8.1	Global weights and the comparators	35
8.2	Method comparison	37
8.3	Expert influence	39
8.4	Leave-one-expert-out	40
8.5	Resampling stability (bootstrap)	41
8.6	Expert disagreement	43
9	Traceability and reproducibility	45
9.1	Source registry	45
9.2	Calculation trace	45
9.3	Run identity	46
9.4	Provenance	46
9.5	Recomputation	47
10	Verification	49
10.1	Golden tests of the methods	49
10.2	Parity of the analyses	50
10.3	Guards against unintended change	50
10.4	Automated test suite	50
10.5	What verification does not cover	50
11	Novelty and comparison with existing software	51
11.1	The research problem	51
11.2	What is new	51
11.2.1	One judgment model for crisp, triangular fuzzy and interval type-2 fuzzy AHP	51
11.2.2	Implementations that were not available as software	51

11.2.3	A method-independent consistency gate	52
11.2.4	Stability analysis over experts	52
11.2.5	Traceability and reproducibility by design	52
11.3	Comparison with existing software	52
11.4	Relation to the definition of a software result	53
12	Limitations	55
12.1	Scope of the input	55
12.2	Methods	55
12.3	Analyses	55
12.4	Platform and distribution	56
A	Scales and constants	57
A.1	Judgment scale	57
A.2	Random index	57
A.3	Triangular fuzzy mapping	57
A.4	Numeric parameters	58
B	Software environment and third-party components	59
B.1	Platform and main components	59
B.2	Third-party licences	59
	References	61

Chapter 1

Introduction

This chapter states the problem that Fuzzy MCDM addresses, what the software does, and how this document is organised.

1.1 Criteria weighting and the choice of method

In a multi-criteria decision the weights of the criteria carry the decision maker's or the experts' priorities. The analytic hierarchy process (AHP) obtains them from pairwise comparisons on the 1 to 9 scale (Saaty, 1977, 1980). Because verbal judgments are vague, the comparisons have been represented by triangular fuzzy numbers (Buckley, 1985) and, to express uncertainty about the membership function itself, by interval type-2 fuzzy numbers (Kahraman et al., 2014).

Many ways of deriving weights from fuzzy comparison matrices are now published. A review by Liu et al. (2020) notes that papers “seldomly set out the relative benefits of each approach so that the choice of the approach seems arbitrary”, and Ahmed and Kilic (2019) show that the algorithms differ in accuracy. In applied studies, transport planning among them, the weighting method is often chosen without checking whether another reasonable method would change the weights or the ranking of the criteria.

Checking this by hand is laborious and error prone: each method has to be implemented from its source, the methods must receive identical judgments under identical rules for inconsistent experts, and every step should be traceable to its source so that a reviewer can check it.

1.2 What the software does

Fuzzy MCDM supports the whole weighting study:

- a study with a goal, main criteria and their subcriteria, and any number of experts;
- pairwise judgments typed in, pasted, or imported from CSV and Excel files;
- a quality review with the consistency ratio of every expert and comparison set;
- five weighting methods and two equal-weight comparators, run on the same judgments;
- local and global weights, rankings, a comparison of the methods, expert disagreement, leave-one-expert-out analysis and bootstrap stability;
- a calculation trace from every weight back to the raw judgments, with the source of every equation;
- exports: an Excel calculation book, publication figures, a PDF report and a run bundle.

The application runs entirely offline. A study is a single file with the extension `.mcdmlab`.

1.3 Scope of this document

Chapter 2 states the design principles and Chapter 3 the architecture. Chapter 4 defines the input model and Chapter 5 the consistency policy shared by all methods. Chapters 6 and 7

describe the methods with their equations and sources. Chapter 8 covers synthesis and the analyses, Chapter 9 traceability and reproducibility, and Chapter 10 verification. Chapter 11 sets out the novelty of the software and compares it with existing tools, and Chapter 12 lists its limitations. The appendices give the scale tables and the software environment.

The installation and the use of the application are described in the separate *Fuzzy MCDM User Guide*.

Chapter 2

Design principles

Five principles shaped the software. They explain several decisions that the later chapters describe in detail.

2.1 One input, many methods

All methods read the same canonical judgments and the same hierarchy (Chapter 4). What differs between methods is fixed in a versioned *method profile*: the scale or the mapping of crisp judgments to fuzzy numbers, the aggregation of experts, the derivation of priorities and the sources. A difference between two weight vectors can therefore be attributed to the methods, not to the data.

2.2 The same rules for every method

The consistency of each expert's judgments is checked once, on the crisp matrix, before any method runs (Chapter 5). An expert whose matrix fails the check is excluded from that comparison set for every method alike. The software asserts at run time that all methods of a run used the same gate.

2.3 Fail closed

A missing judgment, an unmapped value, a non-convergent eigenvector computation or a weight that is not a finite positive number stops the computation with a clear message. The software never substitutes a default value and never repairs a result silently.

2.4 Every number has a source

Each equation, table, scale and constant used by a method is an object in a source registry that links it to a publication and, where it was verified against the full text, to a page, table or equation number. The calculation trace of a run shows every intermediate value with these links (Chapter 9).

2.5 Frozen, versioned science

The method profiles, scales, the random index table and the expected values of the verification tests are frozen. They change only through a new versioned artefact under a recorded decision; an automated check in the build compares the protected files with a reference revision and fails if any of them changed (Chapter 10). Every run records the identities of the profiles it used, and its identity is a hash of its inputs and settings, so a stored result always states exactly how it was computed.

2.6 Offline and private by construction

The application contains no network code, no telemetry and no updater. All data stays in local files chosen by the user. References to publications open in the system web browser only when the user clicks them.

Architecture

This chapter describes the architecture of Fuzzy MCDM: the runtime and its security boundary, the offline design and what enforces it, the package structure, the flow of a run from judgments to stored results, the study container, the persistence and recovery mechanisms, and the admission of method profiles as runnable.

3.1 Runtime

Fuzzy MCDM is a desktop application built with Electron. It has three parts:

- the main process, a Node.js program that owns the window, the native menu and the file dialogs, performs every file access, and starts and supervises every run;
- the renderer, the user interface, which runs sandboxed: the Node.js integration is off, context isolation is on, and a content security policy allows the page to load only itself and grants no network connection;
- the preload bridge, which exposes to the page exactly one typed object and nothing else.

A run never executes in the window process. The main process starts a separate utility process for each run, sends it the current study and the run configuration, and kills the process when the run settles, fails, times out or is cancelled. The interface stays responsive while a run executes, the worker reports its stage and progress back, and a cancelled run can deliver no result. The run is given a time budget of 10 minutes without the bootstrap analysis and 30 minutes with it.

The boundary between the renderer and the host is a typed contract. One interface in the application core package declares every channel together with the types of its arguments and its return value, and the renderer and the main process both compile against it. The preload script builds its object from the channel table at load time, so a channel the interface does not declare cannot be called. On the host side, every channel has a runtime validator that checks each argument (its type, its allowed values, size limits) before the request is served. An unknown channel, a wrong number of arguments, an argument that fails its validator, or a message from any page other than the application's own is refused. File paths for opening and saving come only from the native dialogs, never from the renderer.

3.2 Offline design and its enforcement

The application contains no network code, no telemetry and no updater. Chapter 2 states this as a principle; the code enforces it at four levels.

- The source is tested. An integration test walks the sources of the main process, the preload bridge and the renderer and fails if any file imports a network module of Node.js (net, http, https, dns, tls and the common client libraries) or the network object of Electron, or calls a browser network API (fetch, XMLHttpRequest, WebSocket, sendBeacon, EventSource, service workers). The same test checks the dependency lockfile and the desktop package declaration for updater, crash-reporter and analytics packages.

- The runtime blocks everything. At startup the main process installs a network policy on the session: every request whose URL is not a local resource (the `file`, `app` or `devtools` schemes) is cancelled and counted, and every permission request is denied. Window opening is denied, navigation is prevented, and the packaged executable has the fuses set that disable running as plain Node.js, the Node options environment variable, and the inspector command-line arguments, and that load the application only from its packaged archive after checking the archive's integrity.
- The end-to-end suite proves it. An end-to-end test drives the whole workflow, from importing judgments to reading the results, with a deliberately dead proxy, and asserts that the blocked-request counter of the network policy is zero and that no resource of the page points to an `http` or `https` URL.
- The release is scanned. A production gate re-checks the dependency lockfile and the packaged archive for the same forbidden modules, and a release scanner looks through the built application for secret-looking strings and for the endpoints of known telemetry services.

The only outbound action the application performs is opening a publication in the system web browser. It happens only when the user clicks a reference, and only for a Digital Object Identifier or the source's own `https` address; any other address is refused by the host.

No request leaves the computer, and the statement is checked rather than assumed: source tests, the runtime network policy, an end-to-end run under a dead proxy, and the release scanner all fail the build when the boundary is crossed.

3.3 Package structure

The code is organised as a workspace of 11 packages and two applications. Table 3.1 gives the responsibility of each package and its dependencies within the workspace.

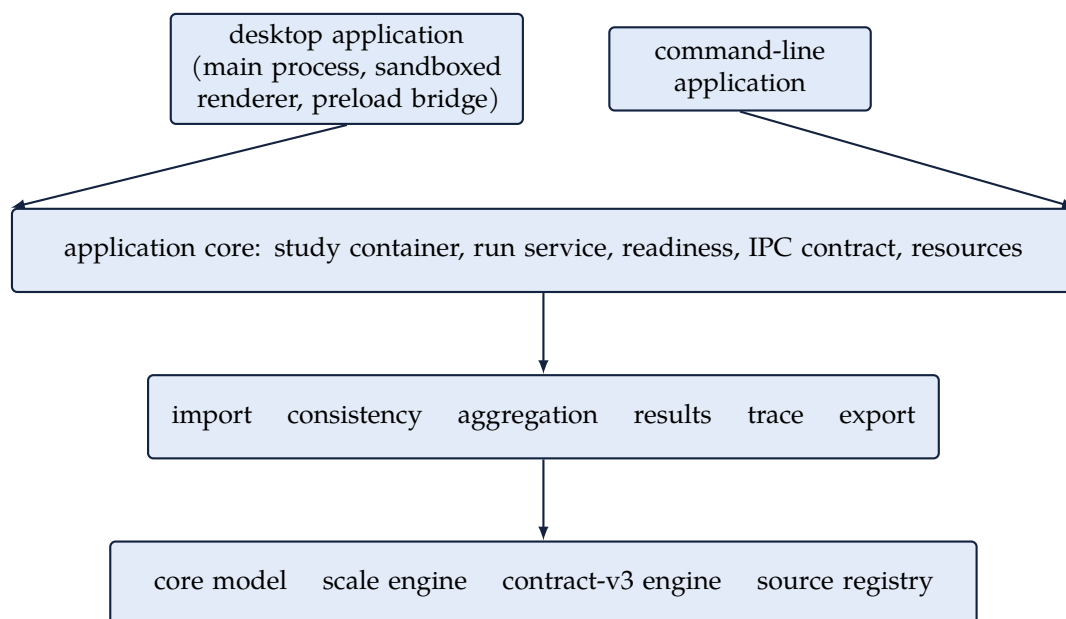


Figure 3.1. The package layers. An arrow points from a package to the packages it uses.

The layers point one way. The foundation row (`core-model`, `scale-engine`, `contract-v3-engine`, `source-registry`) depends on no workspace package other than `core-model`. The

Package	Responsibility	Depends on (workspace)
core-model	Shared study model (project, hierarchy, experts, canonical judgments); deterministic serialization, digests, the input revision	—
scale-engine	Versioned judgment scale profiles and the mappings from crisp judgments to fuzzy numbers	core-model
import-engine	Input adapters (CSV, TSV, XLSX, paste) and the import pipeline to canonical judgments	core-model, scale-engine
consistency-engine	The consistency ratio of the crisp matrices and the random-index profiles	core-model
aggregation-engine	Aggregation of the experts' judgments and the between-expert disagreement	core-model
contract-v3-engine	The frozen computations of the method contract: the five methods, the two comparators, leave-one-expert-out, bootstrap	—
results-engine	Run configuration, run identity, provenance, inclusion, local and global weights, method comparison	core-model, consistency-engine, aggregation-engine
trace-engine	The calculation trace: nodes, builder, integrity check, backward proof	core-model
source-registry	Source records, scientific objects and locators; validation of the registry and of the method profiles	—
export-engine	The calculation book, the run bundle, table serializers, references and figure rendering	core-model, source-registry
app-core	The application core: the study	all of the above

engines of the middle row depend on the foundation row; the results engine also uses the consistency and aggregation engines of its own row. `app-core` is the single package that combines them. The two applications declare `app-core` as their only workspace dependency: the desktop application, described in Section 3.1, and a command-line application that reuses the same core and the same container format for scripted runs and checks. The scientific packages therefore never see the desktop: they are callable from tests and from the command line without the desktop starting. The reverse direction is closed by the same rule: the renderer and the main process reach the engines only through `app-core`, which re-exports every type they need.

3.4 Data flow of a run

A run takes the current inputs of a study and produces one immutable run record. The path from judgments to stored results is:

1. **Judgments.** The import pipeline of `import-engine` (detect, map, preview, validate, commit) and the manual entry both converge on canonical judgments, the single record type that the engines consume (Chapter 4).
2. **Gate.** A run request is validated at the IPC boundary and checked by the readiness gate, which is computed from the current study state. The same gate drives the interface, so what the user sees as a blocker is what the run enforces.
3. **Identity.** The application computes the input revision, a digest of the analysis-defining state, and the run identity, a digest of the effective configuration plus the input revision. A request whose identity matches a recorded run is answered from the stored record; nothing is recomputed.
4. **Computation.** The main process starts the utility process and sends it the study payload (manifest, project, scale profile, judgments) and the location of the bundled resources. The worker loads the resources and the run service executes the run. The consistency of every expert and sibling matrix is computed once, and the inclusion decisions follow (Chapter 5). For each selected method, the contract engine maps the judgments according to the method profile, aggregates the experts, derives the local weights and synthesizes the global weights (Chapters 6 and 7); it then performs the requested analyses (leave-one-expert-out, bootstrap) and the method comparison. Alongside the computation, the trace engine records every operation as a trace node, and the integrity of the trace is asserted before the record is accepted.
5. **Record.** The results package computes the provenance: the run identity, the hashes, the versions, the platform, the threshold actually used, the inclusion lists and the output hash. The record embeds a snapshot of the resources the run used (the method profiles, the source registry, the scale profile, the verification reports) and an immutable copy of the inputs (the input snapshot).
6. **Storage.** The worker returns the record; the main process attaches it to the current study, marks the study dirty, and the record enters the container under `runs/<runId>/. A stored run is immutable: an identical record for an existing run id is accepted without change, and a conflicting one is refused.`
7. **Export.** The calculation book, the run bundle, the tables, the references and the figures are built from the stored record by `export-engine` (Chapter 9).

Since the earlier methods were retired by a recorded decision, the contract engine is the only computation path of a run: every scientific value of a run comes from it.

3.5 The study container

A study is a single file with the extension `.mcdmlab`. The file is a ZIP archive of JSON documents, and the same content always produces the same bytes: the entries are written in sorted order, the JSON is serialized canonically, and every entry carries a fixed modification time.

Entry	Contents
<code>manifest.json</code>	Format version, application and engine versions, git commit, project ID, creation and save times, entry count
<code>project.json</code>	Goal, criteria hierarchy, experts
<code>scale/scale-profile.json</code>	The versioned judgment scale of the study
<code>judgments/canonical.json</code>	All canonical judgments of the study
<code>judgments/imports.json</code>	One record per import: source type, digest of the source, mapping summary, the decisions the researcher made
<code>presentation/figures.json</code>	Optional; the saved figure settings, present only while at least one figure has some
<code>judgments/raw/</code>	Optional; the raw imported files the researcher chose to retain
<code>runs/<runId>/</code>	The run record: <code>run.json</code> (effective configuration), <code>provenance.json</code> , <code>results.json</code> , <code>trace.json</code> , an optional <code>input-snapshot.json</code> , and under <code>snapshots/</code> the copies of the method profiles, the source registry, the scale profile and the verification reports the run used
<code>MANIFEST.sha256</code>	One line per other entry: its SHA-256 digest and its path, in the <code>sha256sum -c</code> line format

Table 3.2. The entries of the study container.

When a study is opened, the codec verifies every entry against `MANIFEST.sha256` and reports the missing, extra and mismatched entries. If any entry fails, the study is opened read-only: reading and export still work, but saving and updating are refused until the user writes a fresh copy with save as. Each run record is checked the same way: its required entries must be present, the configuration recorded in the run and in its provenance must agree, and the provenance must match the method generation. A run record that fails these checks is corrupt and blocks the open.

Runs made with a method that a recorded decision has removed from the product are not read on open. They are left out of the container with a notice that names the run and its methods; the judgments and every other run are untouched, and the study is marked as changed so that the file on disk changes only when the user saves (Chapter 12).

The container carries a format version, currently 1. A file whose version is newer than the application's is refused. A file of an older version is migrated in memory before it is opened: the present migration chain adds the import record and the scale-profile entry that the first format versions lacked and raises the version. A rule enforced on every migration step forbids adding, changing or removing an entry of `runs/`, so a migration can never touch a stored result. A migrated study shows **Converted from an older format – save to update** in the header until the user saves; that save leaves the bytes of the old format beside the file with the extension `.bak`. Forward compatibility is not claimed: the optional entries (the input snapshot, the figure settings) may appear in a file of the same version, an older application can still open such a file, but its save would drop the new entries.

The container is the only storage of a study. There is no database and no copy elsewhere on the machine besides the per-user recovery area of Section 3.6: everything the application knows about a study is in the file.

3.6 Persistence and recovery

Every write of a study file is atomic: the new bytes go to a temporary file in the same directory, are flushed to disk, and are renamed over the target only when the write succeeded. A save that overwrites an existing file leaves the previous bytes beside it with the extension `.bak`.

Recovery copies. In addition to the study file, the application keeps a recovery copy of the current study in the per-user application-data directory, whose location is fixed so that a rename of the product cannot move the data. The copies are listed in a small catalog. A recovery copy is written on a timer the user sets between 30 and 600 seconds (default 120); the writes are serialized through one queue so they never race, and a revision is acknowledged as protected only after its recovery copy or the study file durably holds it. The window header states the real persistence state: **Saved** while the file holds the current revision, **Unsaved changes • recovery copy kept** (or **Not saved to a file • recovery copy kept**) while only the recovery copy does, and **Recovery copy failed** when a write has failed. When a study is opened and its recovery copy is newer than the file, the application offers the two versions and the user chooses; they are never merged silently. The welcome screen lists the studies that have unsaved work.

Changed on disk. Before a save, the host compares the study file with the fingerprint (last modification time and size) it recorded when it last read or wrote the file. If the file changed in the meantime, the save is refused before any write happens, and the user decides whether to overwrite. The fingerprint is deliberately not a content hash: the container is a ZIP archive, and its bytes may change between saves without a semantic change, which a hash would report as a modification.

One instance per data directory. The application takes a single-instance lock keyed by the user-data directory. A second launch of the same installation quits at once and brings the existing window to the front, so two windows can never hold one study.

Closing. A close first runs a durability barrier: the recovery write of the latest revision is flushed, and the window closes silently only when the persistence state is protected. If the flush fails, the application offers to retry, to save the study to a new file, or to close without saving; the destructive choice is never the default.

3.7 Method profiles and the source registry

A method profile is a versioned JSON document that pins down everything that distinguishes one weighting method from another: the scale and the mapping of the judgments, the aggregation of the experts, the derivation of the priorities, the consistency threshold and the random-index profile, the golden cases, and the attribution of every component to scientific objects of the source registry. Profiles carry operational names such as `AHP_EV_C20261008_3`, whose suffix is the version of the method contract they implement. The application ships the profiles read-only with itself, together with the source registry, the BibTeX database and the verification reports.

The source registry is a versioned document with three kinds of entries. A source is a bibliographic record with a verification state (verified against the full text, or verified at the metadata level). An object is a scientific unit: an equation, a table, a mapping, an operator, a threshold or a procedure. A locator is the equation, table, section or page of a source that supports one or more objects, with its own verification level. The registry shipped with version

1.0.0 holds 24 sources, 42 objects and 51 locators. Every entry is validated when the registry is loaded; a registry that fails validation blocks the start of the application.

A method profile may run only when the computed verification report for it says VERIFIED and the production-eligibility document expects VERIFIED. Every other state is refused with a reason. The `verificationStatus` field inside a profile file is inert: it is parsed, but nothing reads it.

Admission works in two stages. At startup, the resource loader reads the profiles, the scale profiles, the registry and the verification reports, validates each of them, and refuses to start when any of them is missing or inconsistent (fail closed). Then a pure function decides which contract methods are runnable: for every profile named in the production-eligibility document, it compares the computed verification report (source coverage, locator coverage, mapping integrity and the golden tests, recorded per profile) with the expected state in that document. A profile is runnable when both are VERIFIED. A profile whose full-text check is pending is blocked with the source that is missing named in the reason, and a profile whose computed state disagrees with the expected one is blocked until the review settles it. A build gate recomputes the reports and fails the build when a computed state differs from the expected one, so the two documents cannot drift apart silently.

Each run stores copies of the profiles, the registry, the scale profile and the verification reports it used inside its record (Chapter 9). A stored result therefore states exactly which versions of the science produced it, and the references of an old run render from its own frozen registry, never from the live one.

Input model

This chapter defines the input that every method reads: the hierarchy of the study, the experts and their influence weights, and the pairwise judgments in their canonical signed form. It closes with the eligibility rule that every computed weight must satisfy. All methods receive this input unchanged (Chapter 2), so a difference between two weight vectors comes from the method, not from the data.

4.1 The study hierarchy

A study has a goal, main criteria under the goal, and optionally subcriteria under each main criterion. The experts compare the members of one *comparison set* at a time: the main criteria form one set, and the subcriteria of one main criterion form another. The size of a comparison set is n . Inactive criteria are ignored; every set holds the active criteria only.

The engine calls each comparison set a *block* and gives it an identifier: MAIN for the goal block and $f_0, f_1, \dots$ for the family blocks. The active criteria under the goal must form one of two shapes:

- FLAT: every active child of the goal is a leaf. The engine sees a single synthetic family f_0 that holds every criterion; the goal block MAIN compares only this family, so it is a 1×1 block whose matrix is (1) for every expert and needs no judgments.
- TWO_LEVEL: every active child of the goal has active children. The goal block MAIN compares the families, and one block per family compares its subcriteria.

Every other shape is rejected with HIERARCHY_INVALID before a run starts: a leaf criterion next to a group, or active children at a third level. The methods support at most two criterion levels; a deeper hierarchy is not flattened, split or silently truncated. A family with a single active child is a 1×1 block and needs no judgments.

A comparison set holds at most ten criteria. The reason is the consistency policy of Chapter 5: the random index table ends at $n = 10$, so the consistency ratio of a larger set is undefined. A group with more than ten active criteria blocks the run (SIBLING_GROUP_TOO_LARGE); the engine itself rejects a larger block with UNSUPPORTED_SIZE.

The engine does not use the identifiers of the application. Families are numbered $f_0, f_1, \dots$ in goal order, and criteria are numbered $c_0, c_1, \dots$ across the families in run order. These synthetic identifiers keep the application's identifiers out of the engine's records.

The hierarchy fixes how local weights combine into global weights. With w_f^{MAIN} the local weight of family f in the goal block and w_c^f the local weight of criterion c in its family block, the global weight of c is

$$g(c) = w_{f(c)}^{\text{MAIN}} w_c^f, \quad (4.1)$$

one multiplication per criterion. In a flat study the goal block has the single weight 1, so $g(c)$ is the local weight of c in the only family block.

4.2 Experts and influence weights

Every expert e of the study carries a raw influence weight w_e . A raw weight is a finite number with $w_e \geq 0$; an expert with weight 0 contributes nothing, and at least one expert must have a

positive weight. The raw weights enter the run input as they are. The application's readiness check reports an invalid vector as `CONTRACT_INFLUENCE_INVALID` (a negative or non-finite entry, an all-zero vector, or a sum that overflows), and the engine rejects the same vector with `INVALID_INFLUENCE`.

Aggregations never use the raw weights directly. After the consistency gate of Chapter 5 has removed the failing experts of a comparison set, the raw weights of the kept experts are normalised over that set, one division each:

$$\beta_e = \frac{w_e}{\sum_{k \in K} w_k}, \quad e \in K, \quad (4.2)$$

where K is the set of experts kept in that comparison set. With equal raw weights, every kept expert receives $1/|K|$. The normalisation runs only over the kept experts, never over all experts first: normalising first and then removing a failing expert could underflow a small kept weight to zero. Once computed, the shares are never rescaled inside a method. A method that receives β gets the constraint

$$|\sum_e \beta_e - 1| \leq 10^{-12}, \quad (4.3)$$

with every β_e finite and non-negative and at least one of them positive. The tolerance covers the binary64 rounding of shares such as n copies of $1/n$; a vector outside it is rejected with `INVALID_INFLUENCE` and is never repaired.

4.3 Pairwise judgments

An expert records one judgment for each pair of criteria of a comparison set. As entered, a judgment has a direction and an intensity: the row criterion is preferred (`ROW`), the column criterion is preferred (`COLUMN`), or the two are equal (`EQUAL`), with an intensity s from 1 to 9 on the Saaty fundamental scale (Saaty, 1977, 1980).

The canonical form of a judgment is a *signed Saaty level*: an integer from the 17 valid levels

$$S \in \{-9, -8, \dots, -2\} \cup \{1, 2, \dots, 9\}. \quad (4.4)$$

The conversion is fixed: `ROW` with intensity s gives the level $+s$, `COLUMN` with intensity s gives $-s$, and `EQUAL` gives 1. The levels 0 and -1 do not exist, so a column judgment at intensity 1 has no signed level; the readiness check blocks it with `CONTRACT_JUDGMENT_NOT_SIGNED_LEVEL` and asks for the comparison to be recorded as equal.

For one expert and one comparison set, the levels form a square matrix S of size n with three rules: the diagonal is $+1$ (a criterion compared with itself); entry (i, j) with $i < j$ is the level of the expert's judgment of that pair; and the lower triangle follows from the reciprocity rule

$$S_{ji} = \begin{cases} -S_{ij} & \text{if } S_{ij} \neq 1, \\ 1 & \text{if } S_{ij} = 1, \end{cases} \quad (4.5)$$

where the second case exists because -1 is not a level. Judgments typed into the application always produce a reciprocal matrix, because only the upper triangle is entered and the lower triangle is derived; an imported matrix is validated against all three rules.

A judgment or imported cell outside the 17 levels is rejected with `INVALID_SIGNED_LEVEL`; a matrix that is not square, whose diagonal is not $+1$, or whose mirror entries break Equation (4.5) is rejected with `INVALID_MATRIX` or `NOT_RECIPROCAL`. A missing judgment is not tolerated either: the run reads every expected pair of every expert, and one that the study does not hold blocks the run with `RUN_BLOCKED_IMPORT_ISSUES` (the readiness check reports it as `MISSING_JUDGMENTS`).

Fail closed: a missing judgment, a column judgment at intensity 1, a level outside the 17 valid ones, or a non-reciprocal imported matrix blocks the run. Nothing is defaulted, guessed or repaired.

The crisp matrix $A = (a_{ij})$, which the consistency policy of Chapter 5 and the AHP eigenvector method read, converts each signed level with one division:

$$a_{ij} = \begin{cases} S_{ij} & \text{if } S_{ij} \geq 1, \\ 1/|S_{ij}| & \text{if } S_{ij} \leq -2. \end{cases} \quad (4.6)$$

The diagonal maps $1 \mapsto 1$. The fuzzy and interval type-2 fuzzy methods do not read A ; they map the same signed levels through their scale tables, from which the entries of the negative levels follow.

4.4 The signed text grammar of the import

A signed level can also arrive as text, through the import. The grammar is an optional sign followed by exactly one digit 1 to 9: 2, +2 and -2 are the levels +2, +2 and -2. The sign may be the ASCII characters + and - or the Unicode minus sign (U+2212), and surrounding ASCII spaces and tabs are ignored. Nothing else is accepted: 0 cannot match the grammar at all, -1 is rejected as a level, and two digits, a decimal point or a fraction such as 1/2 fail the grammar. Text is never coerced: a cell that fails is rejected with `INVALID_SIGNED_LEVEL`, and when a whole matrix is parsed the error names the cell position.

The grammar maps losslessly onto the frozen unsigned import grammar of the earlier scale: level $+s$ is written s and level $-s$ is written $1/s$, so -2 becomes $1/2$ and -9 becomes $1/9$. Both grammars therefore accept the same matrices, and a signed matrix can be exchanged with the unsigned form without loss.

4.5 Worked example: one expert's judgments

The synthetic study *Urban bus fleet* (*synthetic example*) contains a comparison set of three criteria: Cost, Service quality and Environment. Expert A records the three judgments of Table 4.1. Each judgment favours the row criterion, so each signed level is positive and equals the intensity.

Table 4.1. Expert A's judgments of the bus-fleet example, their signed levels and crisp entries.

Pair	Judgment	S_{ij}	a_{ij}
Cost vs. Service quality	2, Cost preferred	+2	2
Cost vs. Environment	4, Cost preferred	+4	4
Service quality vs. Environment	2, Service quality preferred	+2	2

With the diagonal 1 and the reciprocity rule Equation (4.5), the signed matrix and the crisp matrix of the set are

$$S = \begin{pmatrix} 1 & 2 & 4 \\ -2 & 1 & 2 \\ -4 & -2 & 1 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 2 & 4 \\ 1/2 & 1 & 2 \\ 1/4 & 1/2 & 1 \end{pmatrix}. \quad (4.7)$$

The three judgments are transitive: Cost over Environment is $4 = 2 \cdot 2$, the product of Cost over Service quality and Service quality over Environment. Chapter 5 computes the consistency ratio of this matrix.

4.6 Eligibility of the computed weights

Every weight a method produces must pass one final test, the eligibility gate. A run of a method is eligible only when every local weight of every comparison set and every global weight of Equation (4.1) is a finite number greater than 10^{-12} , the constant `MIN_WEIGHT`.

A weight that is not finite, or not greater than 10^{-12} , makes the whole run of that method `NOT_ELIGIBLE`: the run reports the gate verdict and a reason naming the first offending weight, and it reports no weights at all. Nothing is repaired, no epsilon, no clipping, no absolute value, no redistribution. Each method of a run is judged separately; the others are not affected.

Eligibility is a gate, not a correction. A single weight that is not finite and greater than 10^{-12} removes the whole method run; the software never nudges a weight above the threshold.

Consistency policy

This chapter defines the one consistency gate that every method shares: the principal eigenvalue of each expert's crisp matrix, the consistency index and ratio, the random index table, the acceptance threshold, the rules that apply when an expert fails the check, and the information-only group ratio. The gate runs before any method, on the crisp matrix of Chapter 4, and its verdicts are the same for every method of a run.

5.1 The principal eigenvalue

The gate needs $\lambda_{\max}$, the largest eigenvalue of the crisp matrix A , computed together with the principal right eigenvector by the eigenvector method of Saaty (Saaty, 1977, 1980). The engine uses a power iteration. With $\mathbf{1}$ the all-ones vector, it starts from the uniform vector and repeats

$$w^{(0)} = \frac{1}{n}\mathbf{1}, \quad y = Aw, \quad w' = \frac{y}{\sum_i y_i}. \quad (5.1)$$

Each step multiplies the current vector by A and normalises the result to sum 1. The iteration stops when both tests hold:

1. the vector has settled: $\max_i |w'_i - w_i| \leq 10^{-14}$;
2. the returned w' is an eigenvector to the required accuracy: with $z = Aw'$ and $\lambda_{\max} = \sum_i z_i$, the residual $\|z - \lambda_{\max} w'\|_{\infty} \leq 10^{-12}$, where $\|\cdot\|_{\infty}$ is the maximum absolute component.

Both $\lambda_{\max}$ and the residual are computed from the returned w' . The iteration takes at most 10 000 steps; a matrix that has not converged by then stops the computation with the error `NO_CONVERGENCE`, and no result of a failed solve is ever used. A step whose normalisation sum is not a positive finite number stops with the same error. A 1×1 matrix returns immediately, with $\lambda_{\max}$ equal to its single entry. The solver requires a non-empty square matrix (`INVALID_MATRIX` otherwise) with finite positive entries (`INVALID_VALUE` otherwise); it does not recheck reciprocity, because the signed input was validated before (Chapter 4) and one group matrix it receives is not exactly reciprocal (Section 5.6).

5.2 Consistency index and consistency ratio

For a comparison set of size $n \geq 3$, the consistency index and the consistency ratio are

$$\text{CI} = \frac{\lambda_{\max} - n}{n - 1}, \quad \text{CR} = \frac{\text{CI}}{\text{RI}_n}, \quad (5.2)$$

where RI_n is the average random consistency index of size n (Table 5.1). The equations are Saaty's (Saaty, 1977, 1980); a published statement with worked numbers appears in Tan et al. (2007, formulas (3)–(6), p. 257).

For $n \leq 2$ the engine sets $\text{CI} = \text{CR} = 0$ by rule and accepts the matrix. Nothing is lost by the rule: a reciprocal 2×2 matrix has $\lambda_{\max} = 2$ exactly, so there is no inconsistency to measure.

5.3 The random index table

The average random consistency index RI_n is the mean consistency index of large numbers of randomly filled reciprocal matrices of size n . The engine uses the series of Saaty (1980), verbatim; it stores the entries for $n = 3$ to 10 and applies the $n \leq 2$ rule of the previous section instead (Table 5.1). The series is corroborated in full text by Tan et al. (2007, Table 2, p. 257) for n up to 9 and by Sarçın and Karaca (2023, Table 6), which attributes it to Saaty (1980), for n up to 15.

Table 5.1. Average random consistency index RI of Saaty (1980). For $n \leq 2$ the engine sets $CI = CR = 0$ by rule; the entries shown for $n = 1$ and $n = 2$ are the rule's outcome.

n	1	2	3	4	5	6	7	8	9	10
RI_n	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

The table ends at $n = 10$, and so does the supported size of a comparison set (Chapter 4): a larger matrix is rejected with `UNSUPPORTED_SIZE`, because no random index exists for it.

5.4 The acceptance threshold

A matrix passes the individual gate when

$$CR \leq 0.10, \quad (5.3)$$

and the comparison is inclusive: a matrix with CR exactly 0.10 passes. The threshold is Saaty's (Saaty, 1980). Published applications state the rule as $CR < 0.10$ (Tan et al., 2007, p. 257); the software accepts the boundary value as well.

5.5 The gate and its rules

The gate is the individual crisp Saaty CR of every expert in every comparison set (rule R1). It runs before any method, on the crisp matrix Equation (4.6) of the input model, and it depends only on the matrices, never on the method. The run asserts this at run time: every method must report, for each comparison set, the same kept experts and the same status and CR for every expert, bit for bit, and a method that reports a different gate fails the run.

The rules, as the code names them:

- R1: each expert's matrix in each comparison set is accepted when its CR satisfies Equation (5.3); the verdict is `PASS` or `CR_FAIL`.
- R2: a `CR_FAIL` matrix is reported verbatim and excluded from the aggregation of that one comparison set only. The matrix is never altered, no epsilon, clipping or redistribution. The excluded expert still counts in every other comparison set it passes, and the influence shares are renormalised over the kept experts (Equation (4.2)).
- R5: a run without exclusions has the primary status `PRIMARY_ALL_PASS`. A run with at least one exclusion has the status `PRIMARY_WITH_CR_EXCLUSIONS` and is labelled as not fully validated.
- R6: the group consistency ratio (Section 5.6) is information only; it never gates a run.
- R8: a comparison set with no usable expert stops the run with an error: `ALL_CR_FAIL` when every matrix of the set is `CR_FAIL`, and `NO_KEPT_INFLUENCE` when experts pass but their combined influence weight is 0.

The application's readiness check mirrors the gate on the very input the run would see, before the run starts: it reports `NO_ELIGIBLE_EXPERT_MATRIX` for a comparison set that no expert passes, and `CONTRACT_NO_KEPT_INFLUENCE` for one whose passing experts all have influence weight 0.

Rules R3 and R4 cover an optional diagnostic. On request only, an inclusive run keeps every expert, `CR_FAIL` matrices included, with the influence shares normalised over all experts of the set. Its result is labelled `DIAGNOSTIC_INCLUDES_CR_FAIL`, it is never a primary result, and it is not stored with the run record.

One gate, every method: each expert's crisp Saaty CR, per comparison set, threshold 0.10 inclusive. A failing matrix is excluded from that comparison set only and is never revised.

5.6 Group consistency ratio

After the gate, each method aggregates the kept experts of a comparison set into one group matrix and derives its local weights from it. What group consistency information a run reports is fixed per method family (Table 5.2); it is information only (rule R6) and never a gate.

Table 5.2. Basis of the group consistency ratio, per method family.

Method family	Basis of the group CR
AHP (eigenvector)	Saaty CR of the aggregated group matrix
Fuzzy AHP, triangular (Buckley; FICSM)	none; no group CR is defined
Interval type-2 fuzzy AHP (both variants)	Saaty CR of the DTraT matrix of the aggregate

For the AHP eigenvector method, the aggregate is the influence-weighted geometric mean of the kept experts' crisp matrices, and its Saaty CR is reported. For the two interval type-2 fuzzy methods, the entries of the aggregated matrix are defuzzified with the DTraT measure and the Saaty CR of that matrix is reported; the DTraT matrix is not exactly reciprocal, which the eigen-solver tolerates because it requires only finite positive entries. For the two triangular fuzzy methods no group CR is defined.

5.7 Worked example: consistency of one expert

The crisp matrix of Expert A of the bus-fleet example is Equation (4.7) of Chapter 4,

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 1/2 & 1 & 2 \\ 1/4 & 1/2 & 1 \end{pmatrix}. \quad (5.4)$$

The power iteration starts from $w^{(0)} = (1/3, 1/3, 1/3)$. One step gives $y = (7/3, 7/6, 7/12)$ with $\sum_i y_i = 49/12$, hence

$$w^{(1)} = \left(\frac{4}{7}, \frac{2}{7}, \frac{1}{7}\right) \approx (0.5714, 0.2857, 0.1429). \quad (5.5)$$

The judgments are transitive ($4 = 2 \cdot 2$), so $Aw^{(1)} = 3w^{(1)}$ exactly: the vector is already the eigenvector, the second step confirms it, and the engine returns after two steps with residual 0:

$$\lambda_{\max} = 3, \quad \text{CI} = \frac{3-3}{3-1} = 0, \quad \text{CR} = \frac{0}{0.58} = 0. \quad (5.6)$$

The CR is below the threshold of Equation (5.3), so Expert A passes the gate with status PASS. The vector Equation (5.5) is the principal eigenvector, which the AHP eigenvector method uses as the local weights of the set.

Crisp and triangular fuzzy AHP methods

This chapter specifies the two crisp AHP computations and the two triangular fuzzy AHP methods: for each one its method profile, the scale or mapping it applies, the aggregation of the experts into one group matrix, the derivation, defuzzification and normalisation of the local weights, the group consistency information it reports, the golden cases that verify it, and its sources with their locators. The interval type-2 fuzzy methods follow in Chapter 7. The chapter closes with the four computations on one example.

6.1 Overview

Every method of this chapter receives the same input: the hierarchy, the kept experts of each comparison set with their influence shares β_e (Equation (4.2)), and their judgment matrices in signed form. What differs is fixed in a versioned method profile (Chapter 2): the mapping of the signed levels, the group aggregation, the priority derivation and the sources. Table 6.1 lists the four computations and the role the method contract gives each of them.

Table 6.1. The two crisp AHP computations and the two triangular fuzzy AHP methods of this chapter.

Method	Profile	Role	Classification
AHP (Saaty 1980), eigenvector	AHP_EV_C20261008_3	primary candidate	source-locked
AHP row geometric mean (check)	internal check of the AHP_EV profile	diagnostic only	published components
Fuzzy AHP, Buckley (1985) geometric mean	FAHP_BUCKLEY_C20261008_3	primary candidate	source-locked
Fuzzy AHP, FICSM (Ahmed & Kilic 2019)	FAHP_FICSM_C20261008_3	robustness variant	published components
			published method with documented extension

A primary candidate is one of the methods a run reports as its main results. The robustness variant is offered to show how sensitive the ranking is to the choice of the weighting method. The diagnostic check is internal: the engine implements it and the tests run it, but no run computes or reports it, and it does not appear in the method selection. The comparators beside the methods (equal weights, hierarchical and flat) are not weighting methods and are specified with the analyses.

6.2 The shared geometric group step

All four computations aggregate the kept experts first and derive the priorities from one group matrix; this is the aggregation of individual judgments (AIJ). The aggregation operator is the influence-weighted geometric mean

$$M(x_1, \dots, x_E; \beta) = \exp\left(\sum_{e: \beta_e > 0} \beta_e \ln x_e\right), \quad (6.1)$$

where E is the number of kept experts of the comparison set, x_e one value per expert and β_e the influence shares of Equation (4.2). A term with $\beta_e = 0$ is skipped entirely, so an expert with zero influence contributes nothing. A value x_e that is not finite and positive while $\beta_e > 0$

stops the run with `INVALID_VALUE`; nothing is defaulted. The shares are validated once against Equation (4.3) and are never rescaled inside a method.

The geometric mean is the published way to aggregate ratio judgments: it is characterised for the synthesis of ratio judgements by Aczél and Saaty (1983) and discussed as the AIJ rule by Forman and Peniwati (1998). The method profiles cite it at (Mortazavi & Kuzembayev, 2022, eq. 8, p. 10).

For the crisp computations, each kept matrix is first converted with Equation (4.6), giving one crisp matrix $A^{(e)} = (a_{ij}^{(e)})$ per expert. The group matrix $G = (g_{ij})$ aggregates the upper triangle and derives the rest:

$$g_{ij} = M(a_{ij}^{(1)}, \dots, a_{ij}^{(E)}; \beta) \quad (i < j), \quad g_{ji} = \frac{1}{g_{ij}}, \quad g_{ii} = 1. \quad (6.2)$$

The lower triangle is never aggregated; it is the reciprocal of the upper. Matrices that are not square, not of one common size n , or empty stop the run with `INVALID_MATRIX`. The two fuzzy methods aggregate the components of their fuzzy entries with the same operator Equation (6.1), component by component (Section 6.6).

One group matrix first, priorities second: the experts are aggregated with the influence-weighted geometric mean Equation (6.1), over the upper triangle only; the lower triangle is always derived as the reciprocal, never aggregated.

6.3 AHP, the eigenvector method

The profile `AHP_EV_C20261008_3` names the method *Saaty (1980) - Analytic Hierarchy Process, principal right eigenvector method*. Its role is primary candidate and its classification source-locked published components: scale, eigenvector derivation and consistency equations are fixed to the publications and never adjusted. The profile records two limitations. The method is a single-matrix method in its source; the group step is the AIJ geometric mean, which is published and widely used but not part of the original method. And it is crisp: it represents no linguistic vagueness of the judgments.

The computation runs in three steps. Each kept expert's signed matrix is converted to its crisp matrix (Equation (4.6)). The matrices are aggregated into the group matrix G with Equation (6.2). The local weights are then the principal right eigenvector of G :

$$Gw = \lambda_{\max} w, \quad \sum_{i=1}^n w_i = 1, \quad (6.3)$$

where n is the size of the comparison set and $\lambda_{\max}$ the largest eigenvalue of G . The solver is the power iteration of Chapter 5 (Equation (5.1)): it starts from the uniform vector, stops only when the vector has settled to 10^{-14} and the residual is at most 10^{-12} , takes at most 10 000 steps, and a matrix that has not converged stops the run with `NO_CONVERGENCE`; no result of a failed solve is used. The eigenvector priority derivation is Saaty's (Saaty, 1977, 1980); the book itself was not held in full text, and the consistency equations are corroborated by the full-text worked statements of Tan et al. (2007, formulas (3)–(6), p. 257) (Chapter 5).

The group consistency ratio of this method is the Saaty CR of the AIJ aggregate G , computed by the same eigen-solve and reported for information only, never as a gate (Section 5.6).

Two golden cases verify the method (Chapter 10). The published case `PUB_TAN2007_AHP_EV` is the worked example of Tan et al. (2007): the matrix $\begin{pmatrix} 1 & 5 & 3 \\ 1/5 & 1 & 1/4 \\ 1/3 & 4 & 1 \end{pmatrix}$ on p. 256 with its priority vector $(0.6267, 0.0936, 0.2797)$, $\lambda_{\max} = 3.0858$, $CI = 0.0429$ and $CR = 0.0740$ in Table 3 on

p. 257. The engine must match the expected vector at 10^{-10} and the printed values at 10^{-4} . The synthetic case SYN_HIER3_AHP_EV runs a three-family hierarchy whose expected values come from the reference implementation of the method contract.

6.4 The row geometric mean internal check

On the same group matrix G , the engine also implements the row geometric mean

$$r_i = \left(\prod_{j=1}^n g_{ij} \right)^{1/n}, \quad w_i = \frac{r_i}{\sum_{k=1}^n r_k}, \quad (6.4)$$

the operator Equation (6.1) with equal shares over the n entries of the row. Each ratio is computed directly, never as a product with a reciprocal. The derivation is published as formulas (1)–(2) of Tan et al. (2007).

The profile lists this computation as AHP_RGMM_CHECK, a diagnostic-only internal check of the AHP_EV profile with the display name *Row geometric mean method (Tan et al. 2007 formulas 1-2) - numerical check of AHP*. It is a numerical cross-check of the eigen-solve on the same aggregate: two different algorithms that must agree on the same matrix. It computes no consistency of its own. It runs only in the tests: the application does not offer it in the method selection, and no run computes it or stores its weights. Two golden cases pin it down, PUB_TAN2007_AHP_RGMM_CHECK on the published matrix at 10^{-12} (printed values at $5 \cdot 10^{-5}$) and the synthetic SYN_HIER3_AHP_RGMM_CHECK.

6.5 The triangular fuzzy mapping

Both triangular fuzzy methods map each signed judgment level to a triangular fuzzy number $\tilde{a} = (l, m, u)$ with lower bound l , modal value m and upper bound u . The mapping profile is FAHP_TFN_MAPPING_V1; Table 6.2 gives the levels. A positive level s takes row s of the table; a negative level $-s$ takes the reciprocal

$$\tilde{a}^{-1} = (l, m, u)^{-1} = \left(\frac{1}{u}, \frac{1}{m}, \frac{1}{l} \right), \quad (6.5)$$

so the fuzzy entries of a judgment matrix obey the same reciprocity as the crisp ones. The diagonal entry is $\tilde{1} = (1, 1, 1)$. The appendix collects the same values with the other scales (Appendix A). A level the scale has no entry for stops the run with MISSING_SCALE_LEVEL; no value is ever substituted (Chapter 2).

Table 6.2. Triangular fuzzy numbers $\tilde{a} = (l, m, u)$ of the judgment levels (Mortazavi & Kuzembayev, 2022, Table 3, p. 8).

Level s	1	2	3	4	5	6	7	8	9
$\tilde{a}$	(1, 1, 1)	(1, 2, 3)	(2, 3, 4)	(3, 4, 5)	(4, 5, 6)	(5, 6, 7)	(6, 7, 8)	(7, 8, 9)	(9, 9, 9)

The modal value of every level equals the level itself, and the modal value of a reciprocal is the crisp reciprocal. The modal components of a fuzzy group matrix are therefore exactly the crisp entries of Equation (6.2); the example at the end of this chapter uses the fact.

6.6 Fuzzy AHP, the Buckley geometric mean

The profile FAHP_BUCKLEY_C20261008_3 names the method *Buckley (1985) - fuzzy AHP, geometric mean method (triangular fuzzy numbers), as transcribed by Mortazavi & Kuzembayev (2022)*. Its

role is primary candidate and its classification source-locked published components. It applies the triangular mapping of Section 6.5.

The group step aggregates the experts component by component: for $i < j$, each of the three components of the group entry is the influence-weighted geometric mean Equation (6.1) of that component over the experts, and the lower triangle is the reciprocal Equation (6.5):

$$\tilde{g}_{ij} = (M(l_{ij}^{(1)}, \dots, l_{ij}^{(E)}; \beta), M(m_{ij}^{(1)}, \dots, m_{ij}^{(E)}; \beta), M(u_{ij}^{(1)}, \dots, u_{ij}^{(E)}; \beta)) \quad (6.6)$$

for $i < j$, with $\tilde{g}_{ji} = \tilde{g}_{ij}^{-1}$ and $\tilde{g}_{ii} = (1, 1, 1)$, where $l_{ij}^{(e)}, m_{ij}^{(e)}$ and $u_{ij}^{(e)}$ are the components of expert e 's fuzzy entry.

The priority derivation is Buckley's fuzzy geometric mean. Each row is reduced to its component-wise geometric mean with equal shares over the n entries,

$$\tilde{R}_i = \left(\prod_{j=1}^n \tilde{g}_{ij} \right)^{1/n} = (R_{iL}, R_{iM}, R_{iU}), \quad (6.7)$$

each component the operator Equation (6.1) applied to that component along the row. The three component sums

$$\tilde{S} = \sum_{i=1}^n \tilde{R}_i = (S_L, S_M, S_U) \quad (6.8)$$

carry the total, and the fuzzy weight of criterion i divides with crossed bounds:

$$\tilde{W}_i = \left(\frac{R_{iL}}{S_U}, \frac{R_{iM}}{S_M}, \frac{R_{iU}}{S_L} \right). \quad (6.9)$$

The lower bound of a weight is divided by the sum of the upper bounds and the upper bound by the sum of the lower bounds, so the fuzzy weight spans the uncertainty of the whole vector, not of one row alone. Every ratio is computed directly, never as a product with a reciprocal. The defuzzification is the centroid of the three components,

$$c_i = \frac{W_{iL} + W_{iM} + W_{iU}}{3}, \quad (6.10)$$

and the crisp local weights are the normalised centroids

$$w_i = \frac{c_i}{\sum_{k=1}^n c_k}. \quad (6.11)$$

No group consistency ratio is defined for this method: the method contract defines none for the triangular fuzzy family, so a run reports none (Section 5.6). The individual gate of Chapter 5 still applies before the aggregation, as it does to every method.

The golden case PUB_MORTAZAVI2022_T5_FAHP_BUCKLEY is the worked example of Mortazavi and Kuzembayev (2022, Tables 5–8, pp. 11–12): the input is the fuzzy matrix of Table 5, the expected weights come from the reference implementation at 10^{-12} , and the values printed in the publication are compared at $5 \cdot 10^{-4}$. The synthetic case SYN_HIER3_FAHP_BUCKLEY runs the three-family hierarchy. Both are listed in Chapter 10.

The profile records the limitation verbatim: Buckley (1985) itself not held: equations from the CC BY transcription, cross-checked against Boender (1989) Table 4. The TFN spread is a fixed scale convention, not elicited uncertainty. The project does not hold Buckley (1985) in full text; the equations are taken from the open-licence transcription of Mortazavi and Kuzembayev (2022), cross-checked against the Buckley row reproduced in (Boender et al., 1989, Table 4), which agrees within 0.007. The centroid defuzzification convention is also reviewed by Liu et al. (2020, sec. 8.1). The spread of the triangular numbers is a fixed convention of the scale, not uncertainty elicited from the experts (Chapter 12).

6.7 Fuzzy AHP, the FICSM robustness variant

The profile FAHP_FICSM_C20261008_3 names the method *Ahmed & Kilic (2019) - Fuzzy Inverse of Column Sum Method (FICSM)*. Its role is robustness variant and its classification a published method with a documented extension. It applies the same triangular mapping of Section 6.5 and the same group step Equation (6.6). The group step is itself the documented extension: the source evaluates single matrices, and the software first builds one common component-wise geometric group matrix over the experts. The method profile records this extension explicitly.

The priority derivation is the fuzzy inverse of the column sum. Each column is summed component by component,

$$\tilde{C}_j = \sum_{i=1}^n \tilde{g}_{ij} = (C_{jL}, C_{jM}, C_{jU}), \quad (6.12)$$

an ordinary addition per component, and the fuzzy weight of criterion j is the fuzzy reciprocal of its column sum,

$$\tilde{W}_j = \tilde{C}_j^{-1} = \left(\frac{1}{C_{jU}}, \frac{1}{C_{jM}}, \frac{1}{C_{jL}} \right), \quad (6.13)$$

the rule Equation (6.5). The centroid Equation (6.10) and the normalisation Equation (6.11) then produce the crisp local weights, exactly as in the Buckley method. No group consistency ratio is defined for this method either (Section 5.6).

Two limitations follow. The method has no published numerical worked example, and its group step is the documented extension above. It is therefore offered as a robustness variant, not as a primary method. Accordingly, the published golden case PUB_MORTAZAVI2022_T5_FAHP_FICSM is derived on the published input: Mortazavi and Kuzembayev (2022, Table 5) supplies the input matrix but not this method's result, so the expected values come from the reference implementation alone, at 10^{-12} . The synthetic case SYN_HIER3_FAHP_FICSM runs the three-family hierarchy.

The FICSM definition is (Ahmed & Kilic, 2019, sec. 2.3, Table 2, p. 114, and sec. 3.2); the method was first proposed in the MSc thesis of (Ahmed, 2014, eqs. (2.46)–(2.47)). The triangular scale is the common one of (Mortazavi & Kuzembayev, 2022, Table 3, p. 8).

6.8 The four methods on one example

The main-criteria set of the bus-fleet example (Chapter 4) compares Cost, Service quality and Environment. Three experts of equal influence judge it, so every share is $\beta_e = 1/3$; Table 6.3 lists the judgments. Expert A is the expert of Table 4.1.

Table 6.3. The main-criteria judgments of the bus-fleet example.

Pair	Expert A	Expert B	Expert C
Cost vs. Service quality	2, Cost	2, Service quality	3, Cost
Cost vs. Environment	4, Cost	2, Cost	3, Cost
Service quality vs. Environment	2, Service quality	3, Service quality	equal

All three matrices pass the consistency gate of Chapter 5: the consistency ratios are 0.0000 (Expert A), 0.0079 (Expert B) and 0.0000 (Expert C), so every expert is kept in the aggregation. With equal shares, Equation (6.2) is the ordinary geometric mean and every aggregate entry is the cube root of the product of the three experts' entries, for example $g_{12} = \sqrt[3]{2 \cdot \frac{1}{2} \cdot 3} = \sqrt[3]{3}$.

The group matrix is

$$G = \begin{pmatrix} 1 & \sqrt[3]{3} & \sqrt[3]{24} \\ 1/\sqrt[3]{3} & 1 & \sqrt[3]{6} \\ 1/\sqrt[3]{24} & 1/\sqrt[3]{6} & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 1.4422 & 2.8845 \\ 0.6934 & 1 & 1.8171 \\ 0.3467 & 0.5503 & 1 \end{pmatrix}. \quad (6.14)$$

Its principal eigenvalue is $\lambda_{\max} = 3.0010$, so the group consistency ratio that the eigenvector method reports for information is $CR = 0.0009$. The values below are the weights the engine computes for this input; Table 6.4 rounds them to four decimals.

Table 6.4. Local weights of the bus-fleet main criteria, to four decimals.

Criterion	AHP (eigenvector)	AHP row geometric mean (check)	Fuzzy AHP (Buckley)	Fuzzy AHP (FICSM)
Cost	0.4927	0.4927	0.4892	0.4803
Service quality	0.3309	0.3309	0.3292	0.3347
Environment	0.1764	0.1764	0.1816	0.1850

Four observations. All four computations rank Cost first, Service quality second and Environment third; the weight of Cost spans 0.4803 to 0.4927 across the methods. The check column equals the eigenvector column: on a reciprocal 3×3 matrix the normalised row geometric mean is the principal eigenvector, so the two computations agree to every digit shown, which is exactly what the check is there to confirm. The Buckley column is close to the crisp columns because the modal values of its fuzzy weights Equation (6.9) are the crisp geometric-mean weights (0.4927, 0.3309, 0.1764), a consequence of the modal value of every level being the level itself (Section 6.5); the crossed division Equation (6.9) and the centroid Equation (6.10) then shift weight from the strongest criterion to the weakest, from 0.4927 to 0.4892 for Cost and from 0.1764 to 0.1816 for Environment. The FICSM column shifts in the same direction and slightly further, to 0.4803 for Cost, because its column-sum inversion weights the spread of the whole matrix differently than the row means do.

Interval type-2 fuzzy AHP methods

This chapter specifies the two interval type-2 fuzzy AHP methods: for each one its method profile, the nine-level trapezoidal scale it applies with the two membership functions of every level, the aggregation of the experts into one interval type-2 group matrix, the row geometric mean, the interval type-2 sum and division, the defuzzification and normalisation of the local weights, the group consistency information it reports, the golden cases that verify it, and its sources with their locators. An interval type-2 fuzzy number is written $\tilde{\tilde{a}}$ and is a pair of trapezoids. The chapter closes with the check of the defuzzification against the published example of Kahraman et al. (2014).

7.1 Overview

The two methods are one computation with a different scale. Both receive the same input as the methods of Chapter 6: the hierarchy, the kept experts of each comparison set with their influence shares β_e (Equation (4.2)), and their signed judgment matrices. Both map a signed level to an interval type-2 fuzzy number, aggregate the experts into one group matrix, derive fuzzy weights from it, and defuzzify and normalise them. What differs is the nine-level scale, fixed in a versioned mapping profile (Chapter 2). Table 7.1 lists the two computations and the role the method contract gives each of them.

Table 7.1. The two interval type-2 fuzzy AHP methods.

Method	Profile	Role	Classification
IT2 fuzzy AHP, full 1–9 composition (Kahraman et al. (2014) based)	IT2FAHP_FULLL9_BG_KAHRAMAN_C20261008_3	primary benchmark candidate	source-locked published components
IT2 fuzzy AHP, historical frozen (Milošević 2019 scale)	IT2FAHP_LEGACY_R1_C20261008_3	historical frozen reference	source-locked published components

A primary benchmark candidate is offered as one of the methods a run reports as its main results. The historical frozen reference is kept unchanged as a frozen scale for continuity with earlier results; it is not one of the primary candidates. The two comparators beside the methods (equal weights, hierarchical and flat) are not weighting methods and are specified with the analyses.

7.2 The interval type-2 number and the shared operators

An interval type-2 fuzzy number (IT2 number) $\tilde{\tilde{a}}$ is a pair of type-1 trapezoids: an upper membership function $\tilde{a}^U$ and a lower membership function $\tilde{a}^L$. Each trapezoid has four *abscissae* $a_1 \leq a_2 \leq a_3 \leq a_4$ and two *heights* H_1 and H_2 , where H_1 is the height at a_2 and H_2 the height at a_3 :

$$\tilde{\tilde{a}} = (\tilde{a}^U, \tilde{a}^L), \quad \tilde{a}^U = (a_1^U, a_2^U, a_3^U, a_4^U; H_1^U, H_2^U), \quad (7.1)$$

and the same form for the lower part with superscript L. Every operator below acts on the four abscissae of a part and carries its two heights, so a rule written for $(a_1, a_2, a_3, a_4; H_1, H_2)$ applies to the upper and the lower part in the same way. A negative signed level is the reciprocal of its

positive level, the diagonal of a judgment matrix is the scale's level-1 number, and the lower triangle of an aggregate is the reciprocal of the upper triangle, never a second aggregation.

7.2.1 Reciprocal

The reciprocal of an IT2 number reverses the four abscissae of each part and keeps the two heights (Equation (7.2)). This is eq. (31) of Kahraman et al. (2014, eqs. (31)–(32), p. 54).

$$\tilde{a}^{-1} = (\tilde{a}^{U-1}, \tilde{a}^{L-1}), \quad (a_1, a_2, a_3, a_4; H_1, H_2)^{-1} = \left(\frac{1}{a_4}, \frac{1}{a_3}, \frac{1}{a_2}, \frac{1}{a_1}; H_1, H_2 \right). \quad (7.2)$$

The two parts are reciprocal separately, so a reciprocal never mixes the upper and the lower membership function.

7.2.2 Group aggregation

Both methods first aggregate the kept experts of a comparison set into one group matrix; this is the aggregation of individual judgments. For $i < j$ each abscissa of each part is the influence-weighted geometric mean Equation (6.1) of that abscissa over the experts, and each height is the minimum over the experts with $\beta_e > 0$:

$$g_r^P = M(a_r^{P,(1)}, \dots, a_r^{P,(E)}; \beta), \quad H_q^P = \min_{e: \beta_e > 0} H_q^{P,(e)}, \quad (7.3)$$

for $P \in \{U, L\}$, $r \in \{1, 2, 3, 4\}$ and $q \in \{1, 2\}$, where E is the number of kept experts of the comparison set, M is the operator Equation (6.1), and $a_r^{P,(e)}$ and $H_q^{P,(e)}$ are the abscissae and heights of expert e 's IT2 entry. An expert with $\beta_e = 0$ contributes nothing, also not to the minimum, so the heights never come from an empty set; a vector of shares that is not valid is rejected with INVALID_INFLUENCE. The group entry is $\tilde{g}_{ij} = (\tilde{g}_{ij}^U, \tilde{g}_{ij}^L)$ with the abscissae and heights above, the lower triangle is $\tilde{g}_{ji} = \tilde{g}_{ij}^{-1}$, and the diagonal is the scale's level-1 number. The multi-expert geometric mean with minimum heights is the aggregation of Liu et al. (2020, sec. 5.1.4, eqs. (23)–(25)), which cites Görener et al. (2017).

7.2.3 Row geometric mean

On the group matrix, each row is reduced to its IT2 row geometric mean. Every abscissa is the operator Equation (6.1) of that abscissa along the row with equal shares $1/n$, and every height is the minimum along the row:

$$\tilde{R}_i = \left(\prod_{j=1}^n \tilde{g}_{ij} \right)^{1/n} = (\tilde{R}_i^U, \tilde{R}_i^L), \quad (7.4)$$

where n is the size of the comparison set. The row runs over all n entries, the diagonal included. This is eq. (32) of Kahraman et al. (2014, eq. (32), p. 54).

7.2.4 Sum and division

The IT2 row geometric means are added per part. The abscissae add and the heights are the minimum over the added numbers (Equation (7.5)). This is eq. (4) of Kahraman et al. (2014, eqs. (4), (6), p. 49).

$$\tilde{S} = \sum_{i=1}^n \tilde{R}_i = (\tilde{S}^U, \tilde{S}^L). \quad (7.5)$$

The fuzzy weight of criterion i divides its row geometric mean by that total with crossed bounds: the first abscissa of the quotient is divided by the fourth of the divisor and the fourth by the first, so the quotient spans the uncertainty of the whole vector. Each height of a part of the quotient is the smaller of the two heights of that part (Equation (7.6)). This is the division rule below eq. (33) of Kahraman et al. (2014, eq. (33), p. 55).

$$\tilde{W}_i = \frac{\tilde{R}_i}{\tilde{S}}, \quad \frac{(a_1, a_2, a_3, a_4; H_1^a, H_2^a)}{(b_1, b_2, b_3, b_4; H_1^b, H_2^b)} = \left(\frac{a_1}{b_4}, \frac{a_2}{b_3}, \frac{a_3}{b_2}, \frac{a_4}{b_1}; \min(H_1^a, H_1^b), \min(H_2^a, H_2^b) \right). \quad (7.6)$$

The rule is applied to the upper and the lower part separately. Every ratio is computed directly, never as a product with a reciprocal.

7.2.5 Defuzzification and normalisation

The fuzzy weights are defuzzified with the DTraT measure of Kahraman et al. (2014, eq. (24), p. 51). One part of an IT2 number becomes

$$d(a_1, a_2, a_3, a_4; H_1, H_2) = \frac{(a_4 - a_1) + (H_1 a_2 - a_1) + (H_2 a_3 - a_1)}{4} + a_1, \quad (7.7)$$

and the DTraT of the IT2 number is the mean of its two parts,

$$\text{DTraT}(\tilde{a}) = \frac{d(\tilde{a}^U) + d(\tilde{a}^L)}{2}. \quad (7.8)$$

In Equation (7.7) the height H_1 multiplies a_2 alone and the height H_2 multiplies a_3 alone; the term is $H_1 a_2 - a_1$, not $H_1(a_2 - a_1)$. The two readings coincide only when both heights are 1, which is why the defuzzifier is fixed to Equation (7.7) (Section 7.5). The crisp local weights are the normalised DTraT values

$$w_i = \frac{\text{DTraT}(\tilde{W}_i)}{\sum_{k=1}^n \text{DTraT}(\tilde{W}_k)}. \quad (7.9)$$

7.2.6 Group consistency ratio

A run reports one group consistency information per comparison set. For both interval type-2 methods it is the Saaty CR of the matrix of the DTraT values of the aggregated entries

$$D_{ij} = \text{DTraT}(\tilde{g}_{ij}), \quad (7.10)$$

the diagonal included as computed (Section 5.6 and Table 5.2). The matrix D is not exactly reciprocal, because the DTraT of a reciprocal IT2 pair need not be the crisp reciprocal; the eigen-solver tolerates this, as it requires only finite positive entries. The CR uses the random index (Saaty, 1980) and the threshold 0.10 (Chapter 5). It is information only and never a gate.

Both interval type-2 methods aggregate first, then derive: the experts are combined with the influence-weighted geometric mean of the abscissae and the minimum height over the experts with $\beta_e > 0$; the group CR is the Saaty CR of the DTraT values of the aggregate and never gates a run.

7.3 IT2 fuzzy AHP, the full 1–9 composition

The profile `IT2FAHP_FULL9_BG_KAHRAMAN_C20261008_3` names the method *Interval type-2 fuzzy AHP after Kahraman et al. (2014), with the full 1-9 trapezoidal scale of Beresneva & Gordenko (2018) and multi-expert geometric-mean aggregation after Görener et al. (2017) as reviewed by Liu, Eckert & Earl (2020) – published-component composition* (short name *IT2 fuzzy AHP (Kahraman et al. 2014-based, full-9 composition)*). Its role is primary benchmark candidate and its classification source-locked published components. It applies the mapping profile `IT2_FULL9_BG_MAPPING_C20261008_3` and the shared operators of Section 7.2.

7.3.1 The scale

The scale is the full nine-level trapezoidal table of Table 7.2. The odd levels 1, 3, 5, 7 and 9 are the linguistic levels that also appear in the scale of Kahraman et al. (2014, Table 4, p. 54); the even levels 2, 4, 6 and 8 have no counterpart in that table and are added from the full table of Beresneva and Gordenko (2018, Table 11, p. 263). In the upper membership function both heights are 1. In the lower membership function the heights are 0.8 at levels 2 to 9 and 1 at level 1. The method is a composition of published components, not the reproduction of one paper: the IT2 operators, the reciprocal, the row geometric mean, the fuzzy weight and the DTraT defuzzifier are from Kahraman et al. (2014, eqs. (4), (6), p. 49, eqs. (31)–(33), pp. 54–55, Table 4, p. 54, Tables 5–13, pp. 55–56), the full 1–9 scale including the even levels is from Beresneva and Gordenko (2018, Table 11, p. 263), and the multi-expert geometric mean with minimum heights is from Liu et al. (2020, sec. 5.1.4, eqs. (23)–(25)) citing Görener et al. (2017).

Table 7.2. The full 1–9 interval type-2 trapezoidal scale of the method `IT2FAHP_FULL9_BG_KAHRAMAN`, upper and lower part of every level (Beresneva & Gordenko, 2018, Table 11, p. 263).

Level	Part	a_1	a_2	a_3	a_4	H_1	H_2
1	upper	1	1	1	1	1	1
	lower	1	1	1	1	1	1
2	upper	1	1	3	4	1	1
	lower	1.2	1.2	2.8	3.8	0.8	0.8
3	upper	1	2	4	5	1	1
	lower	1.2	2.2	3.8	4.8	0.8	0.8
4	upper	2	3	5	6	1	1
	lower	2.2	3.2	4.8	5.8	0.8	0.8
5	upper	3	4	6	7	1	1
	lower	3.2	4.2	5.8	6.8	0.8	0.8
6	upper	4	5	7	8	1	1
	lower	4.2	5.2	6.8	7.8	0.8	0.8
7	upper	5	6	8	9	1	1
	lower	5.2	6.2	7.8	8.8	0.8	0.8
8	upper	6	7	8.5	9	1	1
	lower	6.2	7.2	8.3	8.8	0.8	0.8
9	upper	7	8	9	9	1	1
	lower	7.2	8.2	8.8	9	0.8	0.8

A level the scale has no entry for stops the run with `MISSING_SCALE_LEVEL`, the levels 0 and –1 are rejected with `INVALID_SIGNED_LEVEL`, and no value is ever substituted (Chapter 2).

7.3.2 Limitations

The method is a composition of published components; no single paper contains the full algorithm, so it is not the reproduction of one publication. The paper of Görener et al. (2017) is not held and is used through the review of Liu et al. (2020), and the source registry's locator for the even scale levels is still pending. No published worked example exercises the levels 2, 4, 6 and 8 with a stated aggregation rule. These levels are covered by a transcription test that compares every value of the mapping profile with the scale table of the method contract by exact equality, by unit tests of the interval type-2 arithmetic, and by the synthetic case (Chapter 10).

7.3.3 Golden cases

Three golden cases pin the method down (Chapter 10). The published case PUB_KAHRAMAN2014_T5_IT2FAHP_FULLL9_BG_KAHRAMAN is the four-criterion matrix of Kahraman et al. (2014, Table 5, p. 55) in signed form, with one expert and the linguistic levels that the case records as E, SS, FS and VS at the intensities 1, 3, 5 and 7. Its expected local weights for the four criteria are 0.2569, 0.6074, 0.0866 and 0.0491, compared at 10^{-12} . The case PUB_KAHRAMAN2014_FIG2_DTRAT exercises the defuzzifier on the four trapezoid pairs of the published figure (Section 7.5). The synthetic case SYN_HIER3_IT2FAHP_FULLL9_BG_KAHRAMAN runs the three-family hierarchy.

7.4 IT2 fuzzy AHP, the historical frozen reference

The profile IT2FAHP_LEGACY_R1_C20261008_3 names the method *Interval type-2 fuzzy AHP* (Kahraman et al. 2014 operators and DTrAT eq (24)) with the Milošević et al. (2019) Table 3 nine-level scale (short name *IT2 fuzzy AHP, historical frozen* (Milošević 2019 scale)). Its role is historical frozen reference and its classification source-locked published components. It applies the mapping profile IT2FAHP_IT2_MAPPING_V1 and the same operators and group step as Section 7.3; only the scale differs.

7.4.1 The scale

The scale is the nine-level table of Milošević et al. (2019, Table 3, p. 12), given in Table 7.3. For levels 2 to 8 the upper membership function is $(k - 1, k - \frac{1}{2}, k + \frac{1}{2}, k + 1; 1, 1)$ and the lower is $(k - \frac{3}{4}, k - \frac{1}{2}, k + \frac{1}{2}, k + \frac{3}{4}; 0.9, 0.9)$. Level 1 is crisp in both parts and level 9 is crisp at 9. The lower heights are 0.9 at every level. The arithmetic, the group step, the row geometric mean, the sum and division, the DTrAT defuzzifier and the normalisation are the Kahraman operators of Section 7.2; the multi-expert aggregation is the one of Section 7.3, from Liu et al. (2020, sec. 5.1.4, eqs. (23)–(25)) citing Görener et al. (2017).

The source registry also lists Abdullah and Najib (2014) for this scale, checked against its metadata only. The values the application ships are those of Milošević et al. (2019, Table 3, p. 12) in Table 7.3.

7.4.2 Limitations

The method is a historical reference, not a current candidate. It is kept unchanged, so that results computed with it earlier can be reproduced, and its scale is not revised (Chapter 12).

Table 7.3. The nine-level interval type-2 trapezoidal scale of the method IT2FAHP_LEGACY_R1, upper and lower part of every level (Milošević et al., 2019, Table 3, p. 12).

Level	Part	a_1	a_2	a_3	a_4	H_1	H_2
1	upper	1	1	1	1	1	1
	lower	1	1	1	1	0.9	0.9
2	upper	1	1.5	2.5	3	1	1
	lower	1.25	1.5	2.5	2.75	0.9	0.9
3	upper	2	2.5	3.5	4	1	1
	lower	2.25	2.5	3.5	3.75	0.9	0.9
4	upper	3	3.5	4.5	5	1	1
	lower	3.25	3.5	4.5	4.75	0.9	0.9
5	upper	4	4.5	5.5	6	1	1
	lower	4.25	4.5	5.5	5.75	0.9	0.9
6	upper	5	5.5	6.5	7	1	1
	lower	5.25	5.5	6.5	6.75	0.9	0.9
7	upper	6	6.5	7.5	8	1	1
	lower	6.25	6.5	7.5	7.75	0.9	0.9
8	upper	7	7.5	8.5	9	1	1
	lower	7.25	7.5	8.5	8.75	0.9	0.9
9	upper	9	9	9	9	1	1
	lower	9	9	9	9	0.9	0.9

7.4.3 Golden cases

Two golden cases verify the method (Chapter 10). The case PUB_KAHRAMAN2014_T5_IT2FAHP_LEGACY_R1 uses the same published four-criterion matrix of Kahraman et al. (2014, Table 5, p. 55) but not this method's result from the publication, so it is labelled derived on the published input. Its expected local weights for the four criteria are 0.2544, 0.6151, 0.0849 and 0.0456, compared at 10^{-12} . The synthetic case SYN_HIER3_IT2FAHP_LEGACY_R1 runs the three-family hierarchy. Both methods also share the defuzzification check of Section 7.5.

7.5 Check against the published example

The two equations of the defuzzifier, Equation (7.7) and Equation (7.8), are checked directly against the worked values of Fig. 2 of Kahraman et al. (2014, eq. (24), Fig. 2, p. 51). The golden case PUB_KAHRAMAN2014_FIG2_DTRAT lists the four trapezoid pairs of Fig. 2a–d and their expected DTraT values. It is labelled with the full 1–9 method, but it exercises only the defuzzifier, which both methods share.

As one instance, the pair of Fig. 2a is the upper trapezoid (10, 22, 32, 52; 0.85, 1) and the lower (15, 22, 32, 47; 0.4, 0.6). Equation (7.7) gives

$$\begin{aligned}
 d^U &= \frac{(52 - 10) + (0.85 \cdot 22 - 10) + (1 \cdot 32 - 10)}{4} + 10 = 28.175, \\
 d^L &= \frac{(47 - 15) + (0.4 \cdot 22 - 15) + (0.6 \cdot 32 - 15)}{4} + 15 = 22.5.
 \end{aligned} \tag{7.11}$$

and the mean of the two parts is DTraT = (28.175 + 22.5)/2 = 25.3375. Table 7.4 lists the four values of the case. The engine is compared with the expected values at 10^{-12} and with the values printed in the publication at $5 \cdot 10^{-3}$.

The check separates the two readings of the defuzzifier. The reading of eq. (24) used here carries $H_1 a_2 - a_1$ and $H_2 a_3 - a_1$; the reading of Milošević et al. (2019, eq. (11), p. 13) carries $H_1(a_2 - a_1)$ and $H_2(a_3 - a_1)$. The two agree only when both heights are 1, and they differ

Table 7.4. The DTraT values of the golden case PUB_KAHRAMAN2014_FIG2_DTRAT (Kahraman et al., 2014, eq. (24), Fig. 2, p. 51).

Trapezoid pair	DTraT (expected)	DTraT (printed)	Absolute difference
Fig. 2a	25.3375	25.34	0.0025
Fig. 2b	26.2125	26.21	0.0025
Fig. 2c	25.3375	25.34	0.0025
Fig. 2d	23.4625	23.46	0.0025

whenever a height is below 1, so they would give different weights on any matrix with heights below 1. The engine uses Equation (7.7), and the check above is the case that discriminates the two readings.

Synthesis and analyses

This chapter specifies what the software computes once the methods have produced their local weights: the synthesis of local weights into global weights and the two equal-weight comparators, the twelve fields that compare weight vectors, the part that expert influence plays, and three analyses that look at the experts themselves: leave-one-expert-out, the resampling stability of a bootstrap over the experts, and the disagreement between the experts. Every analysis reads the global weights in one fixed order, the leaves of the hierarchy with the families in goal order (Chapter 4), and every analysis works on the experts that passed the consistency gate (Chapter 5) unless the text says otherwise.

Where the method contract defines a rule, the chapter states it. Where the contract is silent, the application adds a rule of its own, and the tables and exports of a run carry a label for that rule in their notes. The kinds of label are listed in Table 8.1; the chapter marks each rule with its label in *italics*.

Table 8.1. The labels that mark the rules of the application and of the method contract.

Label	Meaning
<i>app rule</i>	a rule the application adds where the method contract defines nothing, for example when an analysis cannot be offered or a value is not defined
<i>app reading</i>	the application's reading of a statistic whose definition the contract leaves open
<i>app, documented</i>	a documented choice of the application, here the random generator
<i>method contract definition, not parity-tested</i>	a statistic the contract defines but that is not tested for parity with the reference implementation
<i>app summary, not part of the method contract</i>	the expert disagreement table of Section 8.6
<i>method contract definition</i>	a rule the method contract defines, marked where it could be taken for an application rule
<i>method contract definition, no reference value</i>	a statistic the contract defines but states no reference value for

8.1 Global weights and the comparators

A method returns one vector of local weights per comparison set: the goal block **MAIN** over the families and one block per family over its criteria. The synthesis of Equation (4.1) turns them into global weights. After the last block is done, the engine takes the families in goal order and the criteria of each family in order and computes one multiplication per criterion,

$$g(c) = w_{f(c)}^{\text{MAIN}} w_c^{f(c)}, \quad (8.1)$$

with the symbols of Equation (4.1). The result is a vector of N global weights, one per leaf criterion, in the order just described; N is the number of leaves. Because the local weights of every block sum to 1, the global weights of one family sum to the weight of that family in the

goal block,

$$\sum_{c \in f} g(c) = w_f^{\text{MAIN}} \sum_{c \in f} w_c^f = w_f^{\text{MAIN}}. \quad (8.2)$$

In a flat study the goal block has the single weight 1 and the global weights are the local weights of the only family. Every global weight must also pass the eligibility gate of Chapter 4: a method with a weight that is not finite and greater than 10^{-12} is `NOT_ELIGIBLE`, has no weights at all, and takes part in no comparison and no analysis of this chapter. The tables say so with one row per such method.

The two comparators

Two reference weightings are computed beside the methods. They are comparators, not weighting methods: they read the hierarchy and nothing else, no judgments, no scale and no influence weights, so they carry no information about the experts. Both are exact definitions of the method contract, with no publication behind them, and both are optional in a run. The hierarchy is validated first; the comparator stops with `INVALID_HIERARCHY` when the goal lists no family, when a goal entry has no family block or a family block without criteria, when a family block is not listed in the goal, or when an identifier is used twice. Nothing is repaired.

With F families, n_f criteria in family f and $N = \sum_f n_f$, the hierarchical comparator `EQUAL_HIER` gives each family the weight $1/F$ in the goal block and each criterion $1/n_f$ in its family, and the flat comparator `EW_FLAT` gives every criterion $1/N$:

$$g_{\text{EH}}(c) = \frac{1}{F} \cdot \frac{1}{n_{f(c)}}, \quad g_{\text{EW}}(c) = \frac{1}{N}. \quad (8.3)$$

Table 8.2 lists their roles and limitations.

Table 8.2. The two equal-weight comparators.

Comparator	Short name	Role	Family total and limitation
<code>EQUAL_HIER</code>	Equal weights (hierarchical)	canonical comparator	every family totals $1/F$; no expert information, a comparator and not a weighting method
<code>EW_FLAT</code>	Equal weights (flat)	sensitivity only	family f totals n_f/N , so a family with more criteria than average gains weight and a smaller one loses it; never a primary comparator

The two comparators coincide in a flat hierarchy ($F = 1$, so $g_{\text{EH}}(c) = g_{\text{EW}}(c) = 1/N$), and in every hierarchy whose families all have the same size: the three families of two criteria of the bus-fleet example give $1/6$ to every criterion under both. They differ as soon as the family sizes differ, as the example at the end of Section 8.2 shows. A comparator with equal entries is a constant vector, and a constant vector has no ranking to compare, which is why several comparison fields are empty for it (Table 8.3).

Global weights are one multiplication per criterion, local weight of the family times local weight of the criterion. The comparators read the hierarchy only: they show what equal weights would give, not what the experts said.

8.2 Method comparison

The comparison puts the global weight vectors of the eligible methods and of the selected comparators side by side. Each comparison takes two vectors a and b of N global weights in the order of Section 8.1 and returns exactly twelve fields.

Ranks and ties

Two rankings are used, and the difference matters only when weights tie. The descending rank r_i^a gives rank 1 to the largest weight; equal weights keep their criterion order, so the criterion that comes first in the run order ranks first (*app rule*; the tables carry the caption *ranks: ties go to the earlier criterion*). It is used for the rank-change fields, for the top- k overlaps, and for every rank in the analyses below. The average rank $\bar{r}_i^a$ is the ascending rank in which tied weights share the mean of the ranks they span; it is used for the Spearman correlation only (*ranks: ties share the average rank*). For the weights (0.4, 0.3, 0.3) the descending ranks are (1, 2, 3) and the ascending average ranks are (3, 1.5, 1.5). Ties are detected by exact equality of the stored binary64 values; no tolerance is applied.

The twelve fields

With $\bar{a}$ and $\bar{b}$ the means of the vectors, the three correlations are

$$\rho_S = \text{corr}(\bar{r}^a, \bar{r}^b), \quad (8.4)$$

$$\tau_b = \frac{C - D}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}, \quad (8.5)$$

$$\rho_P = \frac{\sum_i (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_i (a_i - \bar{a})^2 \sum_i (b_i - \bar{b})^2}}, \quad (8.6)$$

where corr is the Pearson correlation of two vectors, so ρ_S (field `spearman`) is the Pearson correlation of the average ranks. In Equation (8.5), $n_0 = N(N-1)/2$ is the number of pairs of criteria, C and D are the numbers of concordant and discordant pairs, n_1 is the number of pairs tied in a and n_2 the number of pairs tied in b ; a pair tied in both vectors is in both counts and in neither C nor D . This is Kendall's τ_b (field `kendall_tau_b`). The field `pearson_descriptive` is ρ_P of Equation (8.6), computed on the weights themselves and not on their ranks; a result outside $[-1, 1]$ through rounding is clipped to that interval. The field name marks it as descriptive.

The five distances and the two rank fields are

$$L_1 = \sum_i |a_i - b_i|, \quad L_2 = \sqrt{\sum_i (a_i - b_i)^2}, \quad \text{TV} = \frac{1}{2}L_1, \quad (8.7)$$

$$D_{\max} = \max_i |a_i - b_i|, \quad D_{\text{fam}} = \max_f \left| \sum_{c \in f} a_c - \sum_{c \in f} b_c \right|, \quad (8.8)$$

$$n_{\Delta} = |\{i : r_i^a \neq r_i^b\}|, \quad \Delta_{\max} = \max_i |r_i^a - r_i^b|, \quad (8.9)$$

the fields `L1`, `L2` and `TV`; `max_abs_diff` and `max_family_abs_diff`; and `n_rank_changes` and `max_rank_change`. For vectors that sum to 1, `TV` is the total variation distance. In D_{fam} the family value is the sum of the global weights of its leaves (*app rule: a family is a level-1 criterion; its value is the sum of its leaves' global weights*); by Equation (8.2) this is the weight of the family in the goal block for a method. A flat hierarchy is one family, so D_{fam} is exactly 0 and the interface adds (*one family*) to the field name (*app rule: a flat hierarchy is one family*,

so $\max_family_abs_diff$ is 0). The last two fields are the top- k overlaps for $k = 3$ and $k = 5$ (top3_overlap , top5_overlap),

$$O_k = \frac{|\{i : r_i^a \leq k \text{ and } r_i^b \leq k\}|}{k}, \quad (8.10)$$

the share of the k leading criteria that the two vectors have in common.

Null rules and checks

Table 8.3 states when a field has no value. A missing value is stored as empty together with its reason; it is never replaced by 0.

Table 8.3. When a comparison field is empty.

Fields	Empty when	Reason shown
spearman, kendall_tau_b, pearson_descriptive	$N < 2$ (<i>app rule: correlations need at least 2 criteria</i>), or either vector is constant	<i>fewer than 2 criteria; a weight vector is constant</i>
top3_overlap, top5_overlap	$N < k$ (<i>app rule: top-k overlap is null when there are fewer than k criteria</i>), or either vector is constant	<i>fewer than k criteria; a weight vector is constant</i>
the five distances	never	—
n_rank_changes, max_rank_change	never, but carry a caveat when either vector is constant	<i>a weight vector is constant</i>

A vector is constant when all its entries are exactly equal. That the correlations and the top- k overlaps are empty for a constant vector is a contract definition (*method contract definition*); the fewer-than- k test comes first, so a top- k field of a study with fewer than k criteria gives the fewer-than- k reason. The two rank-change fields keep their values against a constant vector, because ties against a constant vector are broken by criterion order and say nothing about the weights. They carry the caveat *every criterion ties under equal weights; rank changes against it are not meaningful*, and the caveat follows the constant vector, not the name of the comparator. In a flat study the notes also state that EQUAL_HIER equals EW_FLAT.

The comparison fails closed. A weight that is not finite, vectors of different lengths, an empty vector, or family sizes that are not positive integers adding up to N stop it with INVALID_VALUE or INVALID_HIERARCHY. A zero denominator in τ_b , which only a constant vector could cause, never reaches the formula, because the constant test comes first.

The pair set

A run compares, in this order, every pair of its eligible methods (i, j) with $i < j$, taken in the order of the run's method list, and then every eligible method with every selected comparator. Two comparators are never compared with each other. With M eligible methods and C comparators the table has $M(M - 1)/2 + MC$ rows. The comparison exists when at least two methods are eligible, or when at least one method is eligible and at least one comparator is selected; otherwise the table holds only the note *Needs 2+ methods*. A method that is NOT_ELIGIBLE is in no pair, and its row says *not compared: the method is NOT_ELIGIBLE and is in no pair*, followed by the reason. A comparison always uses the global weights; there is no comparison of local weights.

Worked example: the two comparators

Take a hierarchy of three families with 2, 2 and 3 criteria, so $F = 3$ and $N = 7$. By Equation (8.3) the hierarchical comparator gives $1/6$ to each of the four criteria of the first two families and $1/9$ to each of the three criteria of the third, and the flat comparator gives $1/7$ to every criterion. Table 8.4 shows the comparison of the two.

Table 8.4. EQUAL_HIER against EW_FLAT for families of 2, 2 and 3 criteria.

Field	Exact	Value	Field	Exact	Value
L1	4/21	0.1905	max_family_abs_diff	2/21	0.0952
L2	$1/\sqrt{189}$	0.0727	n_rank_changes	0	0
TV	2/21	0.0952	max_rank_change	0	0
max_abs_diff	2/63	0.0317	correlations, top-k	–	empty

The family totals are $1/3$ for each family under the hierarchical comparator and $2/7, 2/7, 3/7$ under the flat one, so $D_{\text{fam}} = |1/3 - 3/7| = 2/21$, reached in the largest family. The four leaves of weight $1/6$ come before the three of weight $1/9$, and the flat comparator, which ties everywhere, ranks by criterion order, so the two descending rankings are identical and there are no rank changes. That is the case the caveat is for: the zero says only that ties were resolved in criterion order. The correlations and the top- k overlaps are empty because EW_FLAT is constant.

8.3 Expert influence

Each expert e has a raw influence weight $w_e \geq 0$ (Chapter 4). The raw weights are never used directly in an aggregation. The shares used by the group step Equation (6.1) are normalised separately for every comparison set s over the experts kept in that set,

$$\beta_e^s = \frac{w_e}{\sum_{k \in K_s} w_k}, \quad e \in K_s, \quad (8.11)$$

which is Equation (4.2) written for one set, and $\beta_e^s = 0$ for an expert outside K_s . The kept set K_s holds the experts whose matrix in s passed the gate. An expert is therefore kept in one comparison set and excluded in another, and its share differs between the sets. The normalisation runs over the kept experts only, and a share is not rescaled again inside a method.

A set needs at least one kept expert with a raw weight greater than 0. The run stops with ALL_CR_FAIL when every matrix of the set is CR_FAIL, and with NO_KEPT_INFLUENCE when experts pass but all of them have raw weight 0 (Chapter 5). The consistency table of a run lists, for each expert and comparison set, the verdict, whether the expert is kept, and the share β ; a set of size 1 has no matrix and no row.

Equation (8.11) gives each of $|K_s|$ kept experts the share $1/|K_s|$ when the raw weights are equal, which is the rule of the method contract. With unequal raw weights the renormalisation over the kept experts of each set is the application's rule (*app rule for unequal influence: the raw weight is influence times count, and the kept experts' raw weights are renormalised per block*). For illustration, raw weights (2, 1, 1) give the shares (1/2, 1/4, 1/4) in a set where all three are kept, and (0, 1/2, 1/2) in a set where the first expert fails.

The two analyses that change the panel use the raw weights as well. A replicate gives the expert the raw weight $w_e c_e$, where c_e is a non-negative integer count, and the normalisation Equation (8.11) then runs over the kept experts with $c_e > 0$:

$$\beta_e^s = \frac{w_e c_e}{\sum_{k \in K_s} w_k c_k}. \quad (8.12)$$

The consistency gate does not depend on the counts, because it reads the original matrices: the same experts pass or fail in every replicate. An expert with a count above 1 receives a proportionally larger share; its matrix is not duplicated. The comparators (Section 8.1) and the expert disagreement (Section 8.6) ignore influence.

8.4 Leave-one-expert-out

Leave-one-expert-out recomputes every eligible method of the run without one expert at a time and reports how far the weights and the ranking move. With P experts there are P replicates. The count matrix is the $P \times P$ matrix with

$$c_k^{(e)} = \begin{cases} 0 & k = e, \\ 1 & k \neq e, \end{cases} \quad (8.13)$$

so replicate e gives the left-out expert the raw weight 0 and every other expert w_k . The counts are the method contract (*replicate e has count 0 for expert e and 1 for every other expert*); the contract states the replicate at equal influence, and multiplying unequal raw weights by the counts is the application's rule of Section 8.3 (*app rule for unequal influence*). The replicate then runs the unchanged hierarchy of Chapter 5 with the shares Equation (8.12): the gate is applied to the same matrices, the shares are renormalised per comparison set over the remaining kept experts, and the method and the eligibility gate run as for the full panel. With equal raw weights, every expert that remains kept in a set receives the share $1/|K'_s|$, where K'_s is the kept set without the left-out expert.

Availability

Before any replicate runs, the application decides whether the analysis is available, in this order. The study is validated first; an invalid hierarchy, influence vector or matrix count stops it with the error of the corresponding check. Then $P \geq 2$ is required; with fewer experts the analysis is unavailable, with the reason *Needs 2+ experts*. Then every comparison set must have a usable expert, an expert whose matrix passes the gate and whose raw weight is greater than 0; a set without one has no result even for the full panel and stops the analysis with ALL_CR_FAIL or NO_KEPT_INFLUENCE, as in Section 8.3. Last, for every expert and every set, some other usable expert must exist. If an expert is the only usable one in a set, leaving that expert out would leave the set empty, and the analysis is unavailable with the reason *An expert is the only usable one in a block*; the expert and the set are named, the first failure in the order of the experts and then of the sets (MAIN first, then the families in goal order). This is the application's rule (*app rule: leave-one-out is refused when any replicate would leave a block without a kept expert*): the analysis is refused as a whole, never run partly. The readiness check of the application reports it before the run starts.

What is reported

For each eligible method the analysis stores the point estimate, the weights of the full panel, and for each left-out expert the local and the global weights of the replicate. The changes against the point estimate are

$$\Delta_{\text{abs}} = \max_i |x_i^{(-e)} - x_i|, \quad \Delta_{\text{rank}} = |\{i : r_i^{(-e)} \neq r_i\}|, \quad (8.14)$$

where x is a weight vector of the full panel, $x^{(-e)}$ the same vector without expert e , and r the descending ranks of the previous section. They are the fields `max_abs_diff` and `n_rank_changes` of Equations (8.8) and (8.9), computed with one family. They are reported twice: for the global

weights, and for the local weights of each comparison set separately. The tables also list, per criterion, the weight with all experts, the weight without the expert and their absolute difference. The two summary values repeat on every row of a method and expert.

A method that is NOT_ELIGIBLE in a replicate has no result for that replicate only, and its row says why (*app rule: a method that is NOT_ELIGIBLE in a replicate has no result for that replicate only*). A method that is NOT_ELIGIBLE for the full panel has no point estimate and is not analysed.

Leave-one-expert-out is all or nothing: it runs for every expert and every comparison set, or it is refused. It reports how far the weights and the ranks move when one expert is removed, nothing more.

8.5 Resampling stability (bootstrap)

The bootstrap asks how stable the weights are with respect to the panel. It draws B new panels of P experts from the original panel with replacement, runs every eligible method on each, and summarises the results. The application calls it *resampling stability*, and the word is deliberate: the spread it reports is the spread of the weights under resampling of the experts, not a confidence interval.

Configuration and availability

The bootstrap is off by default. When it is on, the run configuration holds three items.

- The number of replicates B , a whole number from 10 to 10 000; the default is 2000. Any other value is rejected with the message *The number of replicates must be a whole number from 10 to 10000*.
- The seed, a whole number from 0 to $2^{64} - 1 = 18446744073709551615$, written in decimal without a sign and without leading zeros. Text that is not canonical is rejected, not normalised, because the seed enters the run identity (Chapter 9).
- The generator, `xoshiro256ss-splitmix64-lemire-1`, recorded with the run. A different generator is a different run, and a stored run whose generator the installed version does not have cannot be recomputed.

The bootstrap is available only with $P \geq 2$ experts and with equal raw influence weights: every w_e identical (*app rule: the bootstrap is unavailable with unequal influence*). The method contract does not define how to resample experts of unequal influence, so the application makes the analysis unavailable then, with the reason *Needs equal influence*. With equal weights, Equation (8.12) reduces to $\beta_e^s = c_e / \sum_{k \in K_s} c_k$.

The generator

The application draws the resampling counts itself, from a seeded generator, so that the same seed gives the same counts on every machine (*app, documented*). The generator is the published `xoshiro256**` generator, whose four 64-bit state words are four successive outputs of the published `splitmix64` generator started at the seed; integers in a range are drawn by the published nearly divisionless rejection method. All arithmetic is exact unsigned 64-bit arithmetic, written below modulo 2^{64} , with $\oplus$ the bitwise exclusive or, $\gg$ and $\ll$ shifts, and rotl_k the rotation to the left by k bits.

1. *State from the seed.* Start with $z_0 = \text{seed}$. For $i = 1, \dots, 4$ set $z_i = z_{i-1} + 0x9e3779b97f4a7c15$ and mix $y \leftarrow (z_i \oplus (z_i \gg 30)) \cdot 0xbf58476d1ce4e5b9$, $y \leftarrow (y \oplus (y \gg 27)) \cdot$

0x94d049bb133111eb, output $y \oplus (y \gg 31)$. The four outputs are the state words s_0, s_1, s_2, s_3 .

2. *Next draw.* The output is $\text{rotl}_7(5s_1) \cdot 9$. The state then changes, in this order: $t \leftarrow s_1 \ll 17$; $s_2 \leftarrow s_2 \oplus s_0$; $s_3 \leftarrow s_3 \oplus s_1$; $s_1 \leftarrow s_1 \oplus s_2$; $s_0 \leftarrow s_0 \oplus s_3$; $s_2 \leftarrow s_2 \oplus t$; $s_3 \leftarrow \text{rotl}_{45}(s_3)$.
3. *An integer below R ($1 \leq R < 2^{64}$).* Draw x , form the 128-bit product $m = xR$, and let $l = m \bmod 2^{64}$. If $l < R$, compute $t = (2^{64} - R) \bmod R$ and redraw while $l < t$. The result is $\lfloor m/2^{64} \rfloor$, uniform on $\{0, \dots, R-1\}$ without bias.

The count matrix has B rows and P columns. Replicate b draws P integers below P , one after the other and replicate after replicate (row-major), and $c_{b,e}$ is the number of times the integer e was drawn. Each row sums to P . The same counts apply to every comparison set, which is why the resampling is called joint: an expert is in or out of the replicate for the whole hierarchy. Each replicate then runs exactly as a leave-one-out replicate does, with the raw weights $w_e c_{b,e}$, the unchanged consistency gate and the eligibility gate. The routine that runs the replicates takes a count matrix as input and never draws one, so the parity tests against the fixtures of the method contract feed it the fixture's own counts; the generator is tested apart, by known-answer vectors (Chapter 10).

The count digest identifies the count matrix. It is the SHA-256 of the text in which each row is its counts joined by commas and ended by a line feed, and it is stored with the run. Recomputing a run regenerates the counts from the recorded seed and compares the digest before it compares the results (Chapter 9). With P experts there are only $\binom{2^P-1}{P}$ distinct count rows, for example 10 for three experts and 3 for two, so with a small panel the replicates repeat.

Replicates without a result

A replicate has no result for any method when a comparison set keeps no usable expert, and it is counted by cause and by the set (*app rule: a replicate in which a block keeps no expert has no result for any method; a replicate in which a method is NOT_ELIGIBLE has no result for that method only*). The causes are NO_KEPT_INFLUENCE, when every passing expert of a set has the count 0 in this replicate and the total influence is therefore 0; ALL_CR_FAIL, when every matrix of a set fails the gate; and INVALID_INFLUENCE, when the influence vector is rejected for a named set. The bootstrap needs a point estimate for the full panel, so a set in which every matrix fails never reaches the replicates; in practice the cause is NO_KEPT_INFLUENCE. A method whose weights fail the eligibility gate in a replicate is NOT_ELIGIBLE for that replicate and method only; it is counted per method. For each method, the replicates that have a result are the sample of its statistics.

Statistics

Let n_r be the number of replicates with a result of a method. The statistics are shown for a method only if $n_r \geq 2$ (*app rule: statistics use the replicates with a result; with fewer than 2 results no statistic is shown*); otherwise the method is marked *too few replicates with a result*, and only the counts of replicates with a result and of NOT_ELIGIBLE replicates are reported. The tables carry the note *conditional on a usable panel*: the statistics describe the replicates in which the panel could be used. They are computed for every global weight and for the local weights of every comparison set. For one criterion, let $x_1, \dots, x_{n_r}$ be its weights in the replicates with a result.

Mean (app reading: arithmetic mean over the replicates with a result) and standard deviation

(method contract definition, not parity-tested, with the divisor $n_r - 1$):

$$\bar{x} = \frac{1}{n_r} \sum_r x_r, \quad s = \sqrt{\frac{1}{n_r - 1} \sum_r (x_r - \bar{x})^2}. \quad (8.15)$$

Median and percentiles (method contract definition, no reference value) use the linear-interpolation definition known as type 7, the default of common numerical libraries. For the sorted values $x_{(1)} \leq \dots \leq x_{(n_r)}$ and a percentile $p \in [0, 100]$, with $h = (n_r - 1)p/100$, $j = \lfloor h \rfloor$ and $t = h - j$,

$$Q(p) = x_{(j+1)} + t(x_{(\min(j+2, n_r))} - x_{(j+1)}). \quad (8.16)$$

The median is $Q(50)$. The engine evaluates the interpolation in the form that the common libraries use, so that the values agree to the last digit; it rejects values that are not finite or not sorted.

Rank statistics. Every replicate vector is ranked by the descending rank of Section 8.2, ties to the earlier criterion, and the same quantities are computed on the ranks of each criterion: the median rank, the same percentiles, the rank distribution (the number of replicates at each rank), the modal rank, the most frequent rank, with the smallest rank on a tie, the share of replicates at rank 1, and the share $P(\text{top } 3)$ of replicates with a rank of at most 3 (*app reading: share of replicates whose rank_desc is at most 3*).

Spearman against the point estimate. For each replicate, the Spearman correlation Equation (8.4) of its global weight vector with the vector of the full panel is computed, and the median and the minimum over the replicates are reported. A replicate whose vector is constant has no Spearman correlation and is skipped (*app reading: a replicate whose global weights are constant has no Spearman against the point estimate and is skipped; the summary covers the others*); the number of replicates that count is recorded. The two values repeat on every row of a method.

Which percentiles the interface shows. The 5th and the 95th percentiles are the range the interface shows, in the wording *range shown: 5 and 95 percentiles, a resampling range, not a confidence interval*. The 2.5th and the 97.5th percentiles are computed for weights and ranks as well and are stored and exported, but the tables hide them at first, together with the mean and the standard deviation.

The 5th to 95th percentile range describes how the weights move when the panel is resampled. It is not a confidence interval of a population parameter, and with few experts it is coarse. The results state the number of replicates, the seed, the generator, the number of experts and the count digest.

A run without the bootstrap has a time limit of 600 s; with the bootstrap the limit is 30 minutes (1800 s).

8.6 Expert disagreement

The disagreement table describes how far the experts differ on each pair of criteria. It is a summary of the application (*app summary, not part of the method contract*) and is stored with every run that has at least two experts. It is computed from the judgments alone: the original crisp ratio of every expert's judgment, with the influence weights ignored and the matrices that failed the consistency gate included. It does not depend on the method.

For one pair, the judgment of expert k has the crisp ratio q_k of Equation (4.6): s when the row criterion is preferred with intensity s , $1/s$ when the column criterion is, and 1 when they are equal. The pair must hold exactly one judgment per expert; a second judgment of the same expert for the pair stops the computation with the error `INVALID_AGGREGATION_INPUT`, and a

ratio that is not finite and positive does as well. With n judgments and $\ell_k = \ln q_k$, the table reports for each pair:

- the number of valid judgments n ;
- the geometric mean, and the geometric standard deviation,

$$\text{GM} = \exp\left(\frac{1}{n} \sum_k \ell_k\right), \quad \text{GSD} = \exp\left(\sqrt{\frac{1}{n-1} \sum_k (\ell_k - \bar{\ell})^2}\right), \quad (8.17)$$

which is empty for $n < 2$;

- the percentage of judgments for the row criterion, for equal, and for the column criterion;
- the median signed intensity, where the signed intensity is $+s$ for the row criterion, $-s$ for the column criterion and 0 for equal (not the signed level of Equation (4.4), in which equal is $+1$), and the median is the middle value, or the mean of the two middle values for an even n ;
- the minimum and the maximum ratio;
- the interquartile range of the log-ratios, the difference between the type-7 percentiles 75 and 25 of ℓ_k (Equation (8.16)). The engine computes it on request only, and the application does not request it, so this column is empty in the stored table.

Rows are sorted by comparison set and then by pair. The table notes record the basis, the semantics EMPIRICAL_BETWEEN_EXPERT_DISAGREEMENT and the inputs ORIGINAL_CRISP_RATIOS.

Because the table ignores both the gate and the influence, a pair can show strong disagreement caused by an expert whom the aggregation excluded or gave a small share. It measures the spread of the opinions, not the treatment of those opinions by a method, and it is not a measure of the quality of the result.

As an example, take the pair Cost versus Service quality of the bus-fleet example (Table 6.3). The three ratios are $q = (2, 1/2, 3)$, since Expert B prefers Service quality with intensity 2. The geometric mean is $\sqrt[3]{2 \cdot \frac{1}{2} \cdot 3} = \sqrt[3]{3} \approx 1.4422$, the entry that the group matrix of Equation (6.14) holds for this pair; the geometric standard deviation is about 2.56. Two of three judgments favour Cost, so the percentages are 66.7, 0 and 33.3; the signed intensities are $(2, -2, 3)$ with median 2; and the ratios range from 0.5 to 3.

Traceability and reproducibility

This chapter describes how Fuzzy MCDM links every result to its inputs and sources: the source registry, the calculation trace, the identity of a run and its provenance record.

9.1 Source registry

The source registry (`references/source-registry.json`, version 1.1.3) lists 24 publications, 42 scientific objects and 51 locators. A scientific object is one equation, table, scale, mapping or constant, for example `RI_TABLE_SAAATY_1980` (the random index) or `V3_IT2_DTRAT_EQ24` (the DTraT defuzzification of Kahraman et al. (2014, eq. 24)). A locator ties an object to a place in a publication: a page, table, equation or section, with a verification state (verified against the full text, pending, or metadata only).

Each method profile names the objects it uses. A method is offered in the application only when its verification record shows that its sources and locators cover its objects, that its mapping is intact and that its golden tests pass (Chapter 10). Some locators are still pending even for verified methods, for example the exact page of the random index in Saaty (1980); Chapter 12 lists them.

9.2 Calculation trace

Every run stores a calculation trace: a directed acyclic graph whose nodes are the steps of the calculation. Each node records

- the operation, one of 24 types such as matrix construction, consistency ratio, group aggregation, row geometric mean, fuzzy weight, defuzzification, normalisation and global weight;
- the run, method, comparison set and, where it applies, the expert;
- references to the nodes or raw judgments it takes as input;
- the formula or table it applies and the scientific objects of the registry behind it;
- its input and output values.

For each comparison set and expert, the trace of a run contains the expert's matrix, its consistency ratio and the gate verdict; then the group aggregation over the kept experts; then the stages of each method; then the local and the global weights. The stages cite the objects named in the method's profile, and building the trace fails if a stage would cite an object the profile does not list.

An integrity check runs before every run is stored. It rejects a trace with a duplicate node, a node of another run, or an input reference that is neither an earlier node nor a raw judgment. A backward proof, run by the integration tests on complete research runs, starts at each final weight and follows the input references down to the raw judgments; it fails if a final weight does not reach a raw judgment, if a cycle exists, or if a scientific step on the path has no scientific object or no formula or table.

The **Calculation trace** page of the application shows the trace from a chosen final result down to the expert matrices, with the equation of each step rendered and the values substituted into it (Figure 9.1).

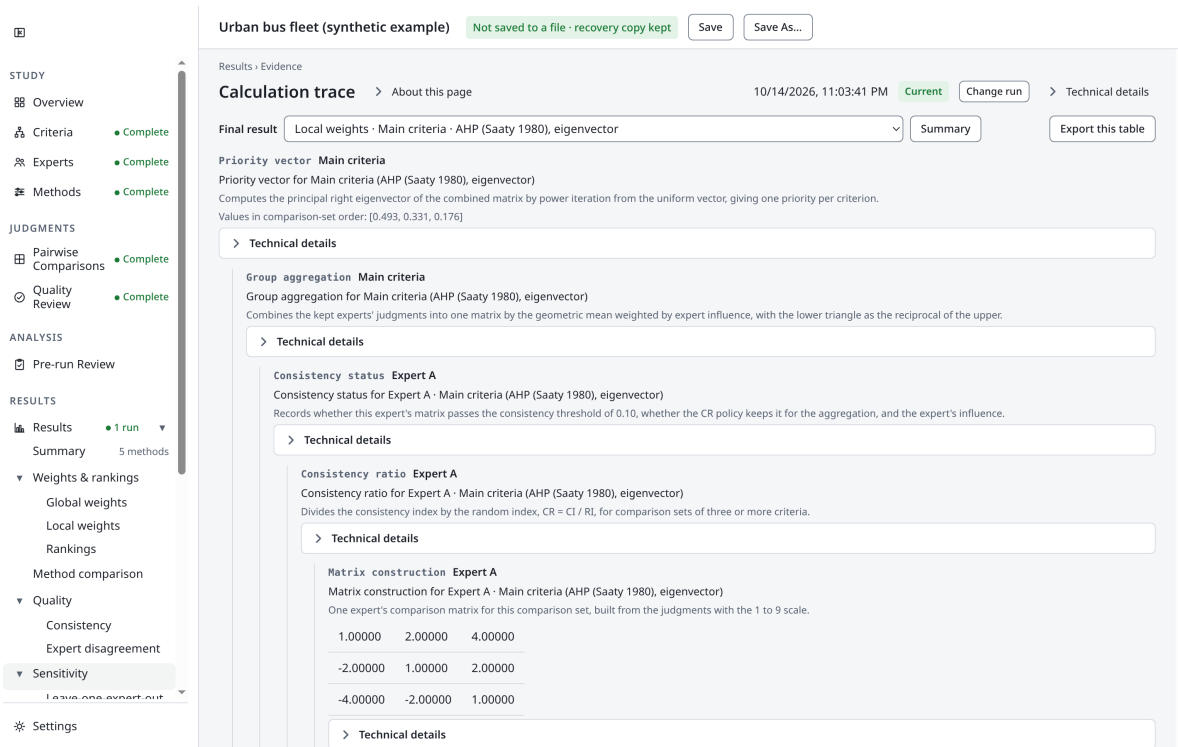


Figure 9.1. The calculation trace of the example study: from the priority vector of the main criteria down to the matrix of one expert. The matrix is shown in signed form, in which $-s$ stands for $1/s$.

9.3 Run identity

A run is identified by content, not by a counter. Two hashes are used, both SHA-256 over a canonical JSON serialisation (keys sorted, no white space, undefined values dropped, non-finite numbers rejected):

Input revision the hash of the analysis-relevant inputs: criteria, experts with their influence weights, the judgments and the scale (version 2 of the input revision, which includes the influence weights).

Run ID the hash of the identity version, the run configuration (method profiles, consistency threshold, aggregation rule, analysis options including the bootstrap replicates, seed and generator) and the input revision.

The same inputs and settings therefore always give the same run ID. A change of any judgment, influence weight, method, option or seed gives a different one. The run bar of the **Results** page compares the input revision of the selected run with that of the current study and labels the run **Current** when they are equal and **Historical** when they differ. A run that records no input revision, from an older version of the application, is labelled **Legacy**.

9.4 Provenance

Each run stores a provenance record with the application and engine versions, the run ID, the input revision, the method profiles and their verification hash, the aggregation rule, and the output hash: the SHA-256 of the sorted method profile identifiers and the results. The run also stores a snapshot of its inputs and copies of the method profiles, scale profile, source registry and verification record it used, so the study file carries everything needed to recompute it.

The provenance is shown on the **Provenance** page, written to the Run_Provenance sheet of

the calculation book, to the appendix of the calculation report and to the run bundle, and, for single-table exports, to a provenance file beside a CSV table or a Provenance sheet in an XLSX table.

9.5 Recomputation

The run service can recompute a stored run from its snapshot: it checks the snapshot against the recorded input revision, checks the bundled profile copies, regenerates the bootstrap counts from the recorded seed and compares their digest, recomputes the run and compares the output hash with the recorded one. The command-line tool used in development exposes this as `mcdmlab reproduce`, which prints `HASH_MATCH` or `HASH_MISMATCH`. The desktop application does not offer recomputation as a command; it shows the fingerprints and whether a run is current, and exports the run bundle with everything a third party needs to repeat the calculation.

Verification

This chapter describes how the implementation of the methods and analyses is checked: golden tests against published worked examples and synthetic cases, parity tests of the analyses, guards against unintended changes to the science, and the automated test suite of the application.

10.1 Golden tests of the methods

Every weighting method is tested against fixed expected values (Table 10.1). Each case states its input matrices in signed form, the expected local weights and, where the case is published, the weights as printed in the publication. The expected values come with the method contract and were computed by an independent reference implementation; the tests compare the engine with them at a tight tolerance and with the printed values at the precision of the publication.

Table 10.1. Golden cases of the weighting methods.

Case and source	Method	Tolerance (expected)	Tolerance (printed)
Tan et al. (2007), matrix on p. 256, Table 3	AHP, eigenvector	10^{-10}	10^{-4}
	row geometric mean (internal check)	10^{-12}	$5 \cdot 10^{-5}$
Mortazavi and Kuzembayev (2022), Table 5	fuzzy AHP, Buckley	10^{-12}	$5 \cdot 10^{-4}$
	FICSM (derived on the published input)	10^{-12}	–
Kahraman et al. (2014), Table 5	IT2 fuzzy AHP, full 1–9	10^{-12}	–
	IT2 fuzzy AHP, historical (derived on the published input)	10^{-12}	–
Kahraman et al. (2014, eq. 24, Fig. 2) Synthetic three-family hierarchy Definition of the comparators	DTraT defuzzification	10^{-12}	$5 \cdot 10^{-3}$
	each of the five methods	10^{-12} ; AHP 10^{-10}	–
	equal weights, hierarchical and flat	10^{-15}	–

Interval type-2 levels 2, 4, 6 and 8 of the full 1–9 scale do not occur in any published worked example with a stated aggregation rule. They are covered by a transcription test that compares every value of the mapping file with the scale table of the method contract (exact equality) and every reciprocal row with the engine’s reciprocal rule, by unit tests of the interval type-2 arithmetic, and by the synthetic cases. The consistency outputs ($\lambda_{\max}$, CI, CR and the gate verdict) are tested in every case at 10^{-10} .

A case is “derived on the published input” when the publication gives the input matrix but not the result of this method; the expected values then come from the reference implementation only.

10.2 Parity of the analyses

The comparison metrics, leave-one-expert-out and the bootstrap statistics are tested against fixtures supplied with the method contract:

- all twelve comparison fields for every pair of methods and every method against each comparator, at 10^{-12} ;
- the leave-one-expert-out weights and changes;
- the bootstrap weight and rank statistics computed on the fixture's own resampling counts, at 10^{-12} .

The reference implementation of the fixtures uses a different random generator, so the generator itself is not parity-tested. It is tested separately: the xoshiro256** generator seeded by splitmix64 reproduces the known answers of the reference C implementation of its authors, and the bounded integer draw is tested for its range and its rejection rule.

10.3 Guards against unintended change

Three guards keep the verified science from changing by accident.

Frozen-file check A build script compares the working tree with a fixed base commit and fails if any file under the frozen paths changed: the method profiles, the scales, the source registry, the references, the golden tests and the earlier scientific engines. Only listed additive edits are allowed. Frozen science changes only by a new versioned artefact under an accepted decision record.

Output baselines The SHA-256 digests of the calculation book, the run bundle, the weights report and every table export of a fixed contract run are recorded. A changed digest fails the tests.

Protected-science baseline A digest of the protected scientific files guards against accidental edits. It is a change detector, not a proof of correctness.

10.4 Automated test suite

Version 1.0.0 passes the type check, the lint rules, the frozen-file check and the test suite of 384 test files with about 4670 tests: unit tests of every package, integration tests of the run service, the study file, the exports and the trace (including the backward proof of Chapter 9), the golden and parity tests above, and 45 end-to-end specifications that drive the application through its interface with Playwright. The privacy tests check that the main and preload processes import no network module, that the renderer calls no network interface, that the content security policy grants no network access, that no updater, crash reporter or analytics package is installed, and that the built application contains no secret-looking string. Equation rendering is tested to work offline.

10.5 What verification does not cover

The tests show that the implementation reproduces its references within the stated tolerances. They do not show that a method is the right choice for a decision problem, and they cannot detect an error that the reference implementation and the transcribed publication share. Locators that are still pending and other limitations are listed in Chapter 12.

Novelty and comparison with existing software

This chapter states what is new in Fuzzy MCDM, compares it with the software found in a search of existing tools, and explains why the result is research software and not an application of known methods with existing tools.

11.1 The research problem

Criteria weighting is the step of a multi-criteria decision in which expert opinion becomes numbers. Since Buckley (1985) the analytic hierarchy process has been extended to fuzzy judgments and, after Kahraman et al. (2014), to interval type-2 fuzzy judgments. Reviews report many published ways of deriving weights from a fuzzy comparison matrix, which “seldomly set out the relative benefits of each approach so that the choice of the approach seems arbitrary” (Liu et al., 2020). Ahmed and Kilic (2019) show that the algorithms differ in accuracy and that the most used one is among the least accurate.

A researcher who wants to know whether a weighting result depends on the method therefore has to implement several methods, apply them to identical judgments under an identical consistency policy, and keep track of which publication defines each step. None of the tools found in the search of Section 11.3 supports this.

11.2 What is new

11.2.1 One judgment model for crisp, triangular fuzzy and interval type-2 fuzzy AHP

All methods read one canonical representation of the experts’ pairwise judgments (signed Saaty levels with reciprocity enforced) and one two-level hierarchy (Chapter 4). The conversion of a crisp judgment into a triangular or an interval type-2 trapezoidal fuzzy number is a separate, versioned mapping per method. Differences between the weight vectors can therefore be attributed to the methods and their scales, not to differences in data entry. This shared design is the basis of the comparison features.

11.2.2 Implementations that were not available as software

- Interval type-2 fuzzy AHP after Kahraman et al. (2014), combined with the full 1 to 9 trapezoidal scale of Beresneva and Gordenko (2018) and multi-expert geometric-mean aggregation as reviewed by Liu et al. (2020). The composition is documented as a composition of published components, not as a reproduction of one paper. The scale levels that no published worked example exercises are covered by transcription, property and independent-reference tests (Chapters 7 and 10).
- Interval type-2 fuzzy AHP with the scale of Milošević et al. (2019), kept frozen as a historical reference.
- Fuzzy AHP with the fuzzy inverse of column sum method (FICSM) of Ahmed and Kilic (2019).

The search found no released implementation of interval type-2 fuzzy AHP and no implementation of FICSM in desktop tools or in Python, R, MATLAB or Julia packages.

11.2.3 A method-independent consistency gate

Each expert's crisp matrix is tested with Saaty's consistency ratio (random index of Saaty (1980), threshold 0.10) before any method runs (Chapter 5). A matrix that fails is excluded from its comparison set for every method alike, the exclusion is recorded, and a run with an exclusion is labelled as not fully validated in its results and exports. Because the gate is the same for every method, a comparison of methods is not confounded by a method-specific treatment of inconsistent experts. When a run is stored, the application checks that all its methods recorded the same gate.

11.2.4 Stability analysis over experts

Besides the method comparison (twelve metrics between every pair of methods and against two equal-weight comparators), the software recomputes all methods without each expert in turn and runs a seeded bootstrap over experts (Chapter 8). The seed, the number of replicates and the generator are part of the run's identity. The results show how strongly a weighting depends on individual experts; the AHP tools reviewed offer sensitivity analysis only as a perturbation of the weights themselves.

11.2.5 Traceability and reproducibility by design

- A source registry links every equation, table, scale and constant used by a method to a publication and, where verified, to its page, table or equation number. The calculation trace of a run shows each intermediate matrix and vector with these references (Chapter 9).
- Every method is fixed by a versioned profile (scale, mapping, aggregation, consistency policy, sources) that is checked against published worked examples, with golden tests transcribed from Tan et al. (2007), Mortazavi and Kuzembayev (2022) and Kahraman et al. (2014), and against synthetic hierarchical cases (Chapter 10).
- A run is identified by the SHA-256 hash of the canonical form of its inputs and settings and stores a hash of its results; the application marks a run whose inputs no longer match the study.
- The method profiles, scales, source registry and golden tests are protected by a frozen-file check in the build, and recorded output baselines guard the engine's results, so a method cannot change silently: a change needs a new versioned profile.

None of the MCDM tools found in the search has these properties: a per-equation source registry, a trace from each judgment to each weight, and a content-hash run identity.

11.3 Comparison with existing software

Table 11.1 summarises a search made on 13 October 2026. It covered official pages, package documentation and source code of the tools listed, and the journals SoftwareX and Journal of Open Source Software. Commercial tools could be assessed only from their public descriptions. The absence of a feature is a search finding, not a proof that no such software exists.

Table 11.1. Capabilities of Fuzzy MCDM and of the closest existing tools found.

Capability	Fuzzy MCDM	Closest existing tools found
Crisp AHP with consistency ratio	yes	common: Expert Choice, Super Decisions, PriEsT (Siraj et al., 2015), the Python package ahpy, the R package ahp
Triangular fuzzy AHP	Buckley and FICSM	Buckley in pyDecision (one matrix); the R package FuzzyAHP with the methods of Krejčí et al. (2017), of Chang, of Wang and Chen, and of Tesfamariam and Sadiq
Interval type-2 fuzzy AHP	two variants	none found; type-2 fuzzy logic toolkits do not implement AHP
Several weighting methods on the same judgments, with comparison metrics	five methods, twelve metrics	PriEsT (crisp derivations from one matrix); a comparison function in the R package FuzzyAHP
One consistency gate for all methods	yes	not found
Leave-one-expert-out and seeded bootstrap over experts	yes	not found
Per-equation source references and a full calculation trace	yes	not found
Content-hash run identity	yes	not found
Offline desktop application	Windows	Super Decisions, Expert Choice Desktop, PriEsT; the libraries are code only

11.4 Relation to the definition of a software result

The national definition of a software result excludes, among others, applications built with known methods and existing tools, added user functions, websites, standard coding and routine debugging. Fuzzy MCDM:

- implements methods that existed only as equations in papers (Section 11.2) and resolves the ambiguities in them by documented decisions recorded in the method contract and the profiles;
- introduces a software design (shared judgment model, method-independent gate, source registry, hash-based run identity) whose purpose is new knowledge about the behaviour of weighting methods;
- was developed by experimental development in the sense of OECD (2015): each method was implemented from its sources, checked against published worked examples, and every ambiguity or discrepancy was resolved by a recorded, versioned decision. Intuitionistic fuzzy AHP, for example, was implemented and then withdrawn from the application when the method contract classified it as research only.

The interface, the packaging and the export formats use standard libraries (Appendix B). They are not claimed as novel.

Limitations

This chapter lists the known limitations of the current version. None of them affects the correctness of the results within the stated scope.

12.1 Scope of the input

- A comparison set holds at most ten criteria, the size covered by the random index table of Saaty (1980); larger sets are rejected.
- Hierarchies have at most two levels: main criteria and subcriteria. Deeper hierarchies are rejected.
- The application computes criteria weights. It does not score or rank alternatives.

12.2 Methods

- The interval type-2 method after Kahraman et al. (2014) with the full 1 to 9 scale is a composition of published components, not the reproduction of one paper. No published worked example exercises the even scale levels with a stated aggregation rule; those parts are covered by transcription, property and independent-reference tests (Chapter 10).
- The spread of the triangular and interval type-2 fuzzy numbers is a fixed convention of each scale. It does not represent uncertainty elicited from the experts.
- A group consistency ratio is not defined for the triangular fuzzy methods; for the interval type-2 methods it is computed on the defuzzified aggregate, which is not exactly reciprocal. Group consistency is reported for information only and does not gate a run.
- Some primary sources were not available in full text (for example the book of Saaty (1980) and the paper of Buckley (1985)); their equations are taken from verified secondary sources, and the source registry marks the locators that are still pending.
- Intuitionistic fuzzy AHP is not offered. Its published variants are classified as research-only by the method contract of the software, and an earlier implementation was withdrawn.

12.3 Analyses

- The bootstrap resamples experts. The 5th to 95th percentile range it reports describes the stability of the weights with respect to the panel, not a confidence interval of a population parameter.
- With few experts the leave-one-expert-out and bootstrap results are coarse; they are reported, not suppressed.

12.4 Platform and distribution

- The application is built for Windows 10 and 11 (x64). The installer is not code-signed, so Windows SmartScreen may warn before the first start.
- There is no automatic update; a new version is installed by running its installer.
- The source code is not distributed.

Appendix A

Scales and constants

This appendix collects the judgment scale, the random index and the triangular fuzzy mapping that the methods use. The values are copied from the scale and mapping profiles that the application ships, which the frozen-file check of Chapter 10 protects. The interval type-2 scales are given with their methods in Chapter 7.

A.1 Judgment scale

Every judgment is a level of the full Saaty 1–9 scale (Saaty, 1980) (profile SAATY_FULL_1_9_V1). A level s in favour of the row criterion gives the matrix entry $a_{ij} = s$; the reciprocal entry is $a_{ji} = 1/a_{ij}$.

Table A.1. The judgment scale.

Level	Label
1	Equal importance
2	Between equal and moderate importance
3	Moderate importance
4	Between moderate and strong importance
5	Strong importance
6	Between strong and very strong importance
7	Very strong importance
8	Between very strong and extreme importance
9	Extreme importance

A.2 Random index

The consistency ratio divides the consistency index by Saaty’s average random index RI_n for a comparison set of n criteria (Chapter 5). The engine holds the values for $n = 3$ to 10; a comparison set may have at most 10 criteria. Sets of one or two criteria are always consistent and have $RI_n = 0$.

Table A.2. Random index RI_n (Saaty, 1980).

n	1	2	3	4	5	6	7	8	9	10
RI_n	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

A matrix is accepted when $CR \leq 0.10$, including a ratio of exactly 0.10.

A.3 Triangular fuzzy mapping

Both triangular fuzzy methods, the geometric mean method (Buckley, 1985) and FICSM (Ahmed & Kilic, 2019), map a level to the triangular fuzzy number $\tilde{a} = (l, m, u)$ of Table A.3

(profile FAHP_TFN_MAPPING_V1). The reciprocal of $\tilde{a} = (l, m, u)$ is $\tilde{a}^{-1} = (1/u, 1/m, 1/l)$.

Table A.3. Triangular fuzzy numbers of the judgment levels.

Level	l	m	u
1	1	1	1
2	1	2	3
3	2	3	4
4	3	4	5
5	4	5	6
6	5	6	7
7	6	7	8
8	7	8	9
9	9	9	9

A.4 Numeric parameters

Table A.4. Numeric parameters of the engine.

Parameter	Value
Consistency threshold	$CR \leq 0.10$
Largest comparison set	10 criteria
Bootstrap replicates B	whole number from 10 to 10 000
Bootstrap seed	whole number from 0 to $2^{64} - 1$
Tolerance for a decimal judgment to match a scale value	10^{-12}
Autosave interval	30 to 600 seconds

Appendix B

Software environment and third-party components

This appendix lists the platform and the main components of version 1.0.0 and summarises the licences of the open-source components that the installer contains.

B.1 Platform and main components

Table B.1. Platform and main components.

Component	Version	Role
Windows	10 or 11, x64	target operating system
Electron	44.3.0	desktop runtime
React	19.3.0	user interface
TypeScript	5.9.3	programming language
KaTeX	0.18.7	equations
ExcelJS	4.4.0	Excel workbooks
PapaParse	5.7.0	reading CSV text
fflate	0.8.3	ZIP archives
Node.js	24	build environment
electron-vite	5	build tool
electron-builder	26.15.3	NSIS installer
Vitest	5	automated tests
Playwright	1.63	end-to-end tests

Only the components above the rule are part of the installed application; the others are used to build and test it.

B.2 Third-party licences

The installer contains 93 open-source packages. Their licences are MIT (70 packages), ISC (13), Apache-2.0 (2), BSD-3-Clause (2), MIT/X11 (2), Unlicense (1), MIT AND Zlib (1) and MIT OR GPL-3.0-or-later (1, used under MIT). One transitive package of the Excel library, `buffers` 0.1.1, states no licence. The Electron runtime and the Chromium components it embeds are distributed under their own licences, which are shipped with the application.

`THIRD_PARTY_NOTICES.md` lists every package with its version, licence and licence text. It is installed with the application and published on the website. The application itself is distributed under the Fuzzy MCDM Research Licence 1.0 (`LICENSE.txt`).

References

- Abdullah, L., & Najib, L. (2014). A new type-2 fuzzy set of linguistic variables for the fuzzy analytic hierarchy process. *Expert Systems with Applications*, 41(7), 3297–3305. <https://doi.org/10.1016/j.eswa.2013.11.028>
- Aczél, J., & Saaty, T. L. (1983). Procedures for synthesizing ratio judgements. *Journal of Mathematical Psychology*, 27(1), 93–102. [https://doi.org/10.1016/0022-2496\(83\)90028-7](https://doi.org/10.1016/0022-2496(83)90028-7)
- Ahmed, F. (2014). Fuzzy analytic hierarchy process: A comparison of the existing algorithms with new proposals [Title page and copyright 2014; the repository record dates the deposit 2015]. <https://research.sabanciuniv.edu/id/eprint/31118/>
- Ahmed, F., & Kilic, K. (2019). Fuzzy analytic hierarchy process: A performance analysis of various algorithms. *Fuzzy Sets and Systems*, 362, 110–128. <https://doi.org/10.1016/j.fss.2018.08.009>
- Beresneva, E., & Gordenko, M. (2018). Applying the methods of system analysis to teaching assistants evaluation. *Proceedings of the Institute for System Programming of the RAS*, 30(3), 251–270. [https://doi.org/10.15514/ISPRAS-2018-30\(3\)-18](https://doi.org/10.15514/ISPRAS-2018-30(3)-18)
- Boender, C. G. E., de Graan, J. G., & Lootsma, F. A. (1989). Multi-criteria decision analysis with fuzzy pairwise comparisons. *Fuzzy Sets and Systems*, 29(2), 133–143. [https://doi.org/10.1016/0165-0114\(89\)90187-5](https://doi.org/10.1016/0165-0114(89)90187-5)
- Buckley, J. J. (1985). Fuzzy hierarchical analysis. *Fuzzy Sets and Systems*, 17(3), 233–247. [https://doi.org/10.1016/0165-0114\(85\)90090-9](https://doi.org/10.1016/0165-0114(85)90090-9)
- Forman, E., & Peniwati, K. (1998). Aggregating individual judgments and priorities with the analytic hierarchy process. *European Journal of Operational Research*, 108(1), 165–169. [https://doi.org/10.1016/S0377-2217\(97\)00244-0](https://doi.org/10.1016/S0377-2217(97)00244-0)
- Görener, A., Ayvaz, B., Kuşakcı, A. O., & Altınok, E. (2017). A hybrid type-2 fuzzy based supplier performance evaluation methodology: The turkish airlines technic case. *Applied Soft Computing*, 56, 436–445. <https://doi.org/10.1016/j.asoc.2017.03.026>
- Kahraman, C., Öztayşi, B., Uçal Sarı, İ., & Turanoğlu, E. (2014). Fuzzy analytic hierarchy process with interval type-2 fuzzy sets. *Knowledge-Based Systems*, 59, 48–57. <https://doi.org/10.1016/j.knosys.2014.02.001>
- Krejčí, J., Pavlačka, O., & Talašová, J. (2017). A fuzzy extension of analytic hierarchy process based on the constrained fuzzy arithmetic. *Fuzzy Optimization and Decision Making*, 16(1), 89–110. <https://doi.org/10.1007/s10700-016-9241-0>
- Liu, Y., Eckert, C. M., & Earl, C. (2020). A review of fuzzy ahp methods for decision-making with subjective judgements. *Expert Systems with Applications*, 161, 113738. <https://doi.org/10.1016/j.eswa.2020.113738>
- Milošević, M. R., Milošević, D. M., Stević, D. M., & Stanojević, A. D. (2019). Smart city: Modeling key indicators in serbia using it2fs. *Sustainability*, 11(13), 3536. <https://doi.org/10.3390/su11133536>
- Mortazavi, A., & Kuzembayev, N. (2022). Development of a weighting procedure for geomechanical risk assessment. *Energies*, 15(18), 6517. <https://doi.org/10.3390/en15186517>
- OECD. (2015). *Frascati manual 2015: Guidelines for collecting and reporting data on research and experimental development*. Paris. <https://doi.org/10.1787/9789264239012-en>
- Saaty, T. L. (1977). A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology*, 15(3), 234–281. [https://doi.org/10.1016/0022-2496\(77\)90033-5](https://doi.org/10.1016/0022-2496(77)90033-5)
- Saaty, T. L. (1980). *The analytic hierarchy process: Planning, priority setting, resource allocation*. McGraw-Hill.

- Sarçın, B., & Karaca, S. (2023). Land suitability assessment for wheat-barley cultivation in a semi-arid region of eastern anatolia in turkey. *PeerJ*, 11, e16396. <https://doi.org/10.7717/peerj.16396>
- Siraj, S., Mikhailov, L., & Keane, J. A. (2015). PriEsT: An interactive decision support tool to estimate priorities from pairwise comparison judgments. *International Transactions in Operational Research*, 22(2), 217–235. <https://doi.org/10.1111/itor.12054>
- Tan, X., Lin, J., Wang, F., Luo, H., Luo, L., & Wu, L. (2007). Evaluation of the effect of a health education campaign of hiv by using an analytical hierarchy process method. *International Journal of Environmental Research and Public Health*, 4(3), 254–259. <https://doi.org/10.3390/ijerph2007030010>