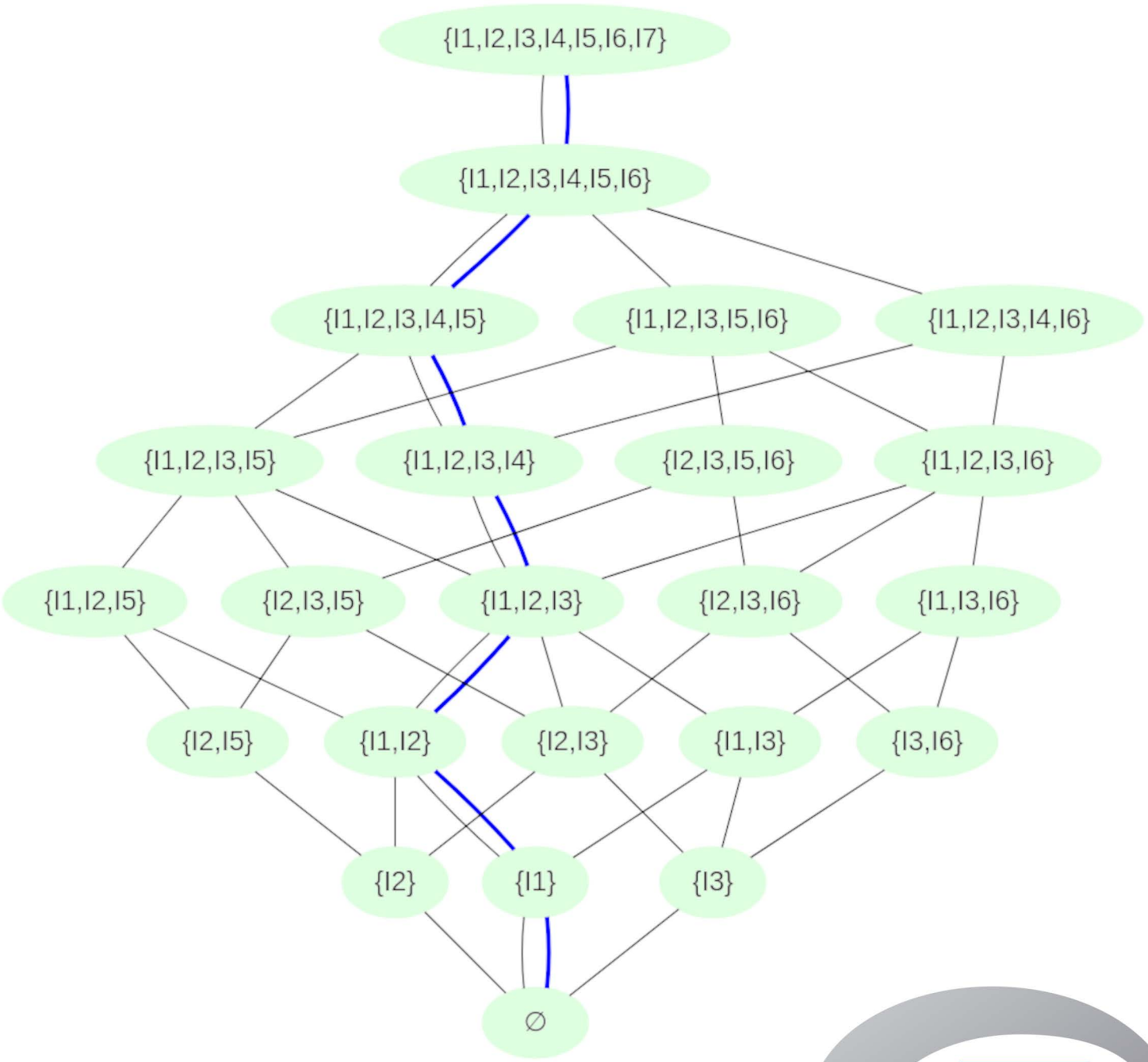

Cord Hockemeyer

R Packages for Knowledge Space Theory

R Packages for Knowledge Space Theory

Version 3.0-3

Cord Hockemeyer (Department of Psychology, University of Graz, Austria)

November 10, 2026

Contents

1 Introduction **2**

2 Knowledge Space Theory **2**
2.1 CbKST . . . 3
2.2 Probabilistic knowledge spaces . . . 4

3 General functionalities **4**
3.1 `kst` . . . 4
3.2 `kstMatrix` . . . 6
3.3 `kstIO` . . . 13

4 KST extensions **13**
4.1 `CbKST` . . . 13
4.2 `pks` . . . 14

5 Specific functionalities **17**
5.1 `CDSS` . . . 17
5.2 `DAKS` . . . 17
5.3 `mycaas` . . . 18

List of described R packages **19**

About this document

- This work is published under the CC-BY-NC-SA license

- To print this document, you may also use its PDF, DOCX, or ODT versions.
- This is a living document; please regard the version number, especially in citations.

1 Introduction

Knowledge space theory (KST; Doignon and Falmagne (1985); Doignon and Falmagne (1999)) offers a set-theoretical framework to structure domains of knowledge through prerequisite relationships. These structures allow an efficient adaptive assessment and training of learners' knowledge.

Since the development of the first R package on KST around 2008 (`kst`; Stahl (2008); Stahl et al. (2026)), a multitude of R packages has been developed which are described in this paper and most of which are available from CRAN. Figure 1 gives an overview of the currently available R packages and their dependencies and connections.

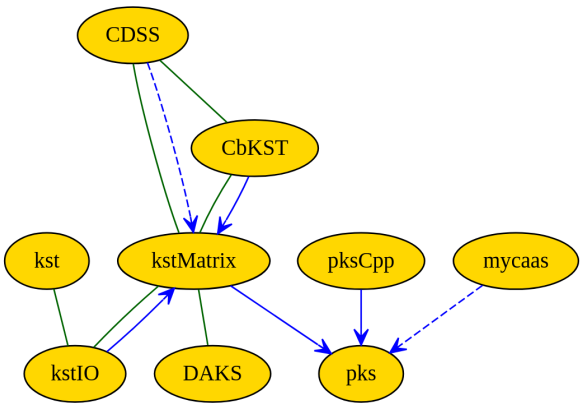

Figure 1: Network of R packages for knowledge space theory

The solid blue arrows depict package imports, the dashed blue arrows package suggestions. The green lines show that packages share S3 object classes.

Chapter 2 gives a short introduction to knowledge space theory (KST). In Chapter 3, packages providing rather general functionalities for basic KST are presented, in Chapter 4, packages for KST extensions follow, namely for competence-based knowledge space theory (CbKST) and for probabilistic knowledge structures. In Chapter 5, packages for more specific purposes are mentioned. Throughout the paper, we will use the `phsg` and `xpl` data sets provided by the `kstMatrix` package.

2 Knowledge Space Theory

Knowledge space theory (KST; Doignon and Falmagne (1985); see also Heller and Stefanutti (2024)) was originally developed as a behavioristic framework supporting efficient adaptive assessment of knowledge where efficiency aims at a reduction of the number of questions to be asked. Opposed to other test-theoretical approaches, KST delivers a non-quantitative result, i.e. the subset of test problems the testee is able to solve instead of a less expressive test score.

Please note that terminology and notations have changed over time, especially in the early years of KST development. In this paper, the line of Doignon and Falmagne (1999) is followed.

In KST, a domain of knowledge is specified as a set Q of dichotomous test items. The *knowledge state* of a person is given by the subset $K \subseteq Q$ of items this person can solve. Usually, not all subsets of Q can be observed as knowledge states because there exist prerequisite relationships between the items. The set $\mathcal{K}$ of all possible knowledge states is called *knowledge structure*. A knowledge structure always contains the emptyset $\emptyset$ and the full item set Q as knowledge states.

A special case of a knowledge structure is the *knowledge space*, a knowledge structure that is closed under union, i.e. for any two knowledge states $K, K' \in \mathcal{K}$, we have $K \cup K' \in \mathcal{K}$. If $\mathcal{K}$ is additionally closed under intersection, it is called *quasi-ordinal knowledge space*.

A knowledge space can also represented by its *basis* $\mathcal{B}$, i.e. the set of all those states which cannot be constructed as union of other, smaller states. The basis is the smallest family of sets from which a knowledge space can be constructed by closure under union. Sometimes, also the term *base* is used for the basis.

Another basic concept in KST is the *surmise relation* $\preceq$. We write $a \preceq b$ for two items $a, b \in Q$ if, from a person mastering item b, we can surmise that this person also masters item a. Sometimes, however, this is not so unique because items may be solvable with different solution paths involving different prerequisites. This is covered by the *surmise function* σ, a generalization of the surmise relation. σ is defined by

$$\sigma : Q \to 2^{2^Q \setminus \emptyset} \setminus \emptyset,$$

i.e. it assigns to each item $q \in Q$ a non-empty set of non-empty subsets $C \subseteq Q$, the *clauses* of q. If a person masters item q, we can surmise that they also master all items in at least one clause of q. If, for all $q \in Q$, we have $|\sigma(q)| = 1$, i.e. each item has exactly one clause, the surmise function can be represented as a surmise relation. We have then $q \preceq q'$ if $q \in C \in \sigma(q') = \{C\}$.

Both, surmise functions and surmise relations have certain properties: reflexivity, transitivity and (for surmise functions) incompatibility, i.e. any two different clauses for an item are not in a subset relation. From mathematical point of view, a surmise relation is a quasi-order.

There is a one-to-one correspondence between the knowledge spaces over a set Q and the surmise functions over Q. Similarly, there is a one-to-one correspondence between the quasi-ordinal knowledge spaces and the surmise relations on Q.

2.1 CbKST

While KST itself is rather behavioristic, there was soon a growing interest of KST researchers in the skills and competencies underlying the observable behavior (see, e.g., Doignon (1994); Düntsch and Gediga (1995); Korossy (1997); Albert and Lukas (1999); Heller and Stefanutti (2024)). Out of the various approaches, that of a *skill map* has prevailed.

A skill map is a triple (Q, S, τ) where Q is a non-empty set of problems, S is a non-empty set of skills, and $\tau : Q \to 2^S \setminus \emptyset$ is a mapping assigning to each problem $q \in Q$ a non-empty set of skills. There are two different interpretations for τ:

- **Conjunctive model:** All skills $s \in \tau(q)$ are necessary for solving item $q \in Q$, and

- **Disjunctive model** Any skill $s \in \tau(q)$ is sufficient for solving item $q \in Q$.

For the conjunctive model, the knowledge state K of a person in the skill state $T \subseteq S$ is delineated by

$$K = \{q \in Q \mid \tau(q) \subseteq T\}.$$

The collection of all knowledge states delineated by some $T \subseteq S$ is the knowledge structure delineated by the skill map (Q, S, τ). In the conjunctive model, this knowledge structure is closed under intersection; in the disjunctive model, the delineated knowledge structure is closed under union.

Like surmise relations in classical KST, skill maps do not provide for test items with different solution paths involving different prerequisites. The solution for this are *skill multimaps*. A skill multimap is a triple (Q, S, μ) where μ is a mapping assigning to each $q \in Q$ a non-empty collection of non-empty subsets of S. The subsets $C \in \mu(q)$ are called *competencies*. For skill multimaps, there is only one interpretation: In order to be able to solve item q, a person must possess all skills $s \in C$ for at least one competency $C \in \mu(q)$.

The knowledge state delineated by a subset $T \subseteq S$ of skills is given by

$$K = \{q \in Q \mid C \subseteq T \text{ for some } C \in \mu(q)\}.$$

2.2 Probabilistic knowledge spaces

Pure KST is deterministic. However, this is unrealistic when regarding students' response behavior to test items. Careless errors and lucky guesses occur. In probabilistic KST, this is treated with the *Basic Local Independence Model* (BLIM). The assumption underlying he BLIM is that this noise depends only on the individual items. Starting with an estimated probability distribution over the knowledge structure (i.e. the probability for each knowledge state to be the learner's state), the probability to observe a response pattern $R \subseteq Q$ given a knowledge state $K \in \mathcal{K}$ is $P(R) = P(K) \cdot P(R|K)$. If β_q denotes the probability for a careless error answering item q and η_q denotes the probability for a lucky guess, this leads to

$$P(R|K) = \prod_{q \in R \cap K} (1 - \beta_q) \cdot \prod_{q \in K \setminus R} \beta_q \cdot \prod_{q \in R \setminus K} \eta_q \cdot \prod_{q \in \bar{R} \cap \bar{K}} (1 - \eta_q).$$

On the one side, the BLIM can be used for probabilistic knowledge assessment (Falmagne and Doignon (1988a)). On the other side, it can also be used for simulating response patterns. The identifiability of the BLIM is a current research issue.

3 General functionalities

3.1 `kst`

The `kst` package (Stahl et al., 2026; Stahl, 2008) is a kind of starting point offering basic functionalities for the work with knowledge spaces. The implementation is insofar close to the theory as it uses the `sets` and `relations` packages by David Mayer. The `kst` package uses S3 classes (i) to make sure the corect object types are passed to the functions and (ii) to be able to use generic methods like `print()` or `plot()`. Figure 2 shows the hierarchy of object classes in the `kst` package.

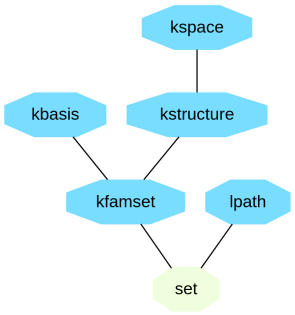

Figure 2: Object classes in the ‘kst‘package

All classes are sub classes of `set`. A core class is `kfamset` which describes a family of sets. Special cases are `kbasis` and `kstructure`, the latter again having a special case in `kspace`. Somewhat separated is the `lpath` class describing a learning path.

Working with `kst`, the first step for us is to convert our example into set notation.

```
kf <- as.famset(phsg$basis)
kf
#> {{"I1"}, {"I2"}, {"I3"}, {"I2", "I5"}, {"I3", "I6"},
#>  {"I1", "I2", "I3", "I4"}, <<set(7)>>}
```

By default, sets with more than five elements are not fully printed as can be seen above. This behavior can be changed by passing a `limit` parameter to the `print()` or `plot()` functions.

```
plot(kf, main="", limit=8)
```

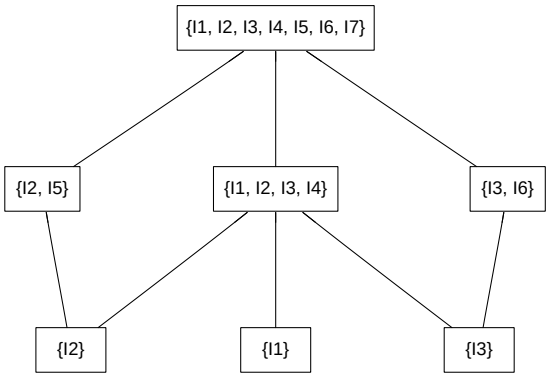

Figure 3: Plotting a family fo sets with the ‘kst‘ package

We can then stepwise derive a knowledge space and, from that, a basis.

```
ks <- kspace(kstructure(kf))
print(ks, limit=8)
#> {{}, {"I1"}, {"I2"}, {"I3"}, {"I1", "I2"}, {"I1",
#>  "I3"}, {"I2", "I3"}, {"I2", "I5"}, {"I3", "I6"},
#>  {"I1", "I2", "I3"}, {"I1", "I2", "I5"}, {"I1", "I3",
#>  "I6"}, {"I2", "I3", "I5"}, {"I2", "I3", "I6"},
#>  {"I1", "I2", "I3", "I4"}, {"I1", "I2", "I3", "I5"},
```

```
#>  {"I1", "I2", "I3", "I6"}, {"I2", "I3", "I5", "I6"},
#>  {"I1", "I2", "I3", "I4", "I5"}, {"I1", "I2", "I3",
#>  "I4", "I6"}, {"I1", "I2", "I3", "I5", "I6"}, {"I1",
#>  "I2", "I3", "I4", "I5", "I6"}, {"I1", "I2", "I3",
#>  "I4", "I5", "I6", "I7"}}
kbase(ks)
#> {{"I1"}, {"I2"}, {"I3"}, {"I2", "I5"}, {"I3", "I6"},
#>  {"I1", "I2", "I3", "I4"}, <<set(7)>>}
```

The `lpath()` function determines all learning paths from the empty set to the full item set as a list of the individual paths. The list is rather long, therefore only the first element is printed.

```
lp <- lpath(ks)
length(lp)
#> [1] 66
print(lp[[1]], limit=8)
#> {{}, {"I1"}, {"I1", "I2"}, {"I1", "I2", "I3"}, {"I1",
#>  "I2", "I3", "I4"}, {"I1", "I2", "I3", "I4", "I5"},
#>  {"I1", "I2", "I3", "I4", "I5", "I6"}, {"I1", "I2",
#>  "I3", "I4", "I5", "I6", "I7"}}
```

`kst` offers several further functions including the computation of neighborhoods and fringes and a deterministic knowledge assessment.

3.2 kstMatrix

Work on the `kstMatrix` package (Hockemeyer et al., 2026; Steiner et al., ress) started originally having a re-implementation of the `kst` package in mind. The underlying idea was that applying a matrix representation should be faster than the set representation because the former is more native to R. Meanwhile, the package has grown considerably beyond that original purpose into a larger set of functions.

Like in `kst`, object classes are used intensively as shown in Figure 4.

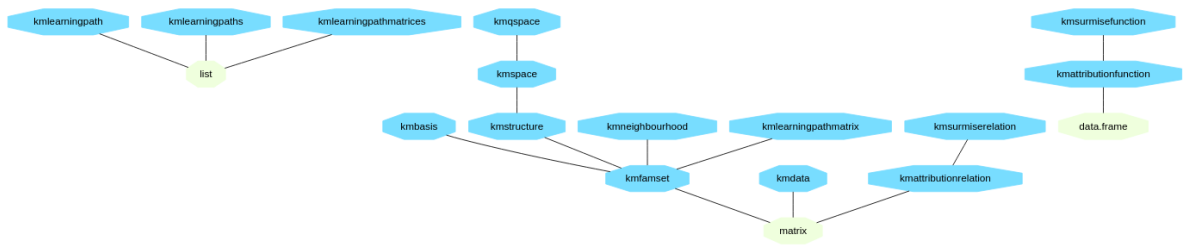

Figure 4: Object classes in the ‘kstMatrix‘ package

Most classes are derived from the `matrix` class. Our list starts with those classes that are similar to the ones from the `kst` package.

- `kmfamset`: Family of sets, subclass of `matrix`
- `kmbasis`: Basis of a knowledge space, subclass of `kmfamset`
- `kmstructure`: Knowledge structure, subclass of `kmfamset`
- `kmspace`: Knowledge space, subclass of `kmstructure`
- `kmqspace`: Quasi-ordinal knowledge space, subclass of `kmspace`
- `kmneighbourhood`: Neighborhood of a state in a knowledge structure, subclass of `kmfamset`
- `kmdata`: Data set, subclass of `matrix`
- `kmattributionrelation`: Attribution relation, subclass of `matrix`
- `kmsurmiserelation`: Surmise relation, subclass of `kmattributionrelation`
- `kmlearningmathmatrix`: Learning path as matrix, subclass of `kmfamset`
- `kmlearningpathmatrices`: List of learning paths in matrix form, subclass of `list`
- `kmlearningpath`: Learning path as list of character strings, subclass of `list`
- `kmlearningpaths`: List of learning paths, subclass of `list`

3.2.1 Basic functions

We start similar as in the description of the `kst` package above.

```
phsg$basis
#>      I1 I2 I3 I4 I5 I6 I7
#> [1,]  1  0  0  0  0  0  0
#> [2,]  0  1  0  0  0  0  0
#> [3,]  0  0  1  0  0  0  0
#> [4,]  1  1  1  1  0  0  0
#> [5,]  0  1  0  0  1  0  0
#> [6,]  0  0  1  0  0  1  0
#> [7,]  1  1  1  1  1  1  1
#> attr(,"class")
#> [1] "kmbasis"  "kmfamset" "matrix"   "array"
kmprettyprint(phsg$basis)
#> [1] "{{I1}, {I2}, {I3}, {I2, I5}, {I3, I6}, {I1, I2, I3, I4}, {I1, I2, I3, I4, I
```

Prettyprinting can also be done in a *simplified* way, i.e. omitting braces etc. However, this is recommended only for single character item names.

```
kmprettyprint(xpl$basis)
#> [1] "{{a}, {b}, {a, c}, {b, c}, {a, b, d}}"
kmprettyprint(xpl$basis, simplified=TRUE)
#> [1] "{a, b, ac, bc, abd}"
```

Plots with `kstMatrix` look by default different depending on the class of the plotted object. Various parameters of vertices and edges can be specified in the function call.

```
plot(phsg$basis)
```

3.2.2 Neighborhoods, fringes, and learning paths

The *n-neighborhood of a knowledge state* is defined as the family of those knowledge states which have a maximal symmetric set difference. By default, the 1-neighborhood is regarded whcih is simply called neighborhood.

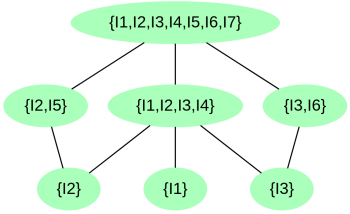

Figure 5: Plotting a basis with ‘kstMatrix‘

```
ksp <- kmspace(phsg$basis)
state = c(1,1,1,1,0,0,0)
kmneighbourhood(state, ksp, include=TRUE)
#>      I1 I2 I3 I4 I5 I6 I7
#> [1,]  1  1  1  0  0  0  0
#> [2,]  1  1  1  1  1  0  0
#> [3,]  1  1  1  1  0  1  0
#> [4,]  1  1  1  1  0  0  0
#> attr(,"class")
#> [1] "kmneighbourhood" "kmfamset"        "matrix"
#> [4] "array"
kmnneighbourhood(state, ksp, distance=2)
#>       I1 I2 I3 I4 I5 I6 I7
#>  [1,]  1  1  0  0  0  0  0
#>  [2,]  1  0  1  0  0  0  0
#>  [3,]  0  1  1  0  0  0  0
#>  [4,]  1  1  1  0  0  0  0
#>  [5,]  1  1  1  0  1  0  0
#>  [6,]  1  1  1  1  1  0  0
#>  [7,]  1  1  1  0  0  1  0
#>  [8,]  1  1  1  1  0  1  0
#>  [9,]  1  1  1  1  1  1  0
#> attr(,"class")
#> [1] "kmneighbourhood" "kmfamset"        "matrix"
#> [4] "array"
```

```
plot(kmneighbourhood(state, ksp, include=TRUE), state=state)
```

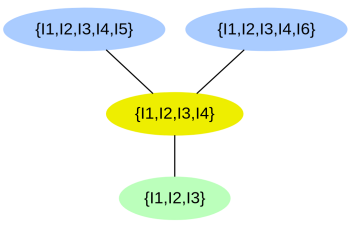

Figure 6: Plot of a neighborhood

Alternatively, a neighborhood an also be marked within the plot of a knowledge structure.

```
plot(ksp, state=state)
```

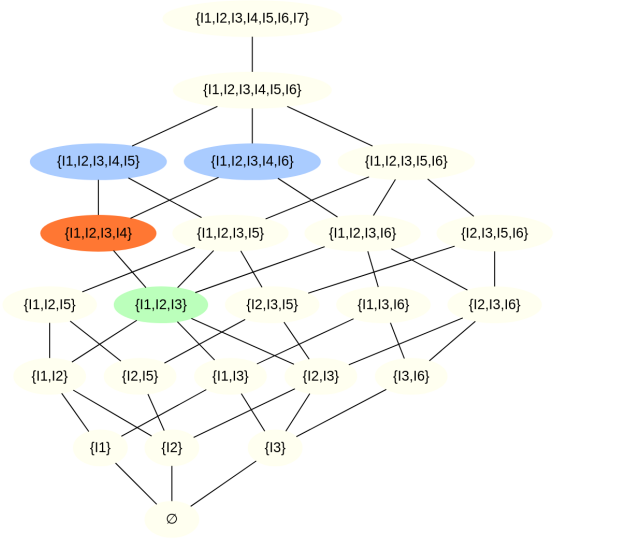

Figure 7: Neighborhood within a knowledge structure

The *fringe* of a knowledge state is the set of items by which the state differs from its 1-neighbors. We distinguish between the *inner fringe* which is part of the state and the *outer fringe* containing the items by which the state can be expanded.

```
kmfringe(state, ksp)
#> I1 I2 I3 I4 I5 I6 I7
#>  0  0  0  1  1  1  0
kminnerfringe(state, ksp)
#> [1] 0 0 0 1 0 0 0
```

Learning paths are ways to get from the empty set (of knowing nothing in the field) to the full item set (of knowing everything). In our example space, we have 66 learning paths.

```
length(kmlearningpaths(ksp))
#> [1] 66
lpm <- kmlearningpathmatrices(ksp)[[1]]
cat(str_wrap(kmprettyprint(lpm), width=60))
#> {} →{I1} →{I1, I2} →{I1, I2, I3} →{I1, I2, I3, I4} →{I1, I2,
#> I3, I4, I5} →{I1, I2, I3, I4, I5, I6} →{I1, I2, I3, I4, I5,
#> I6, I7}
```

```
plot(lpm, structure=ksp)
```

3.2.3 Deriving and combining knowledge structures

`kmunion()` and `kmintersection()` determine the union and intersection, respectively, of knowledge structures. Please note that the union of knowledge spaces corresponds to

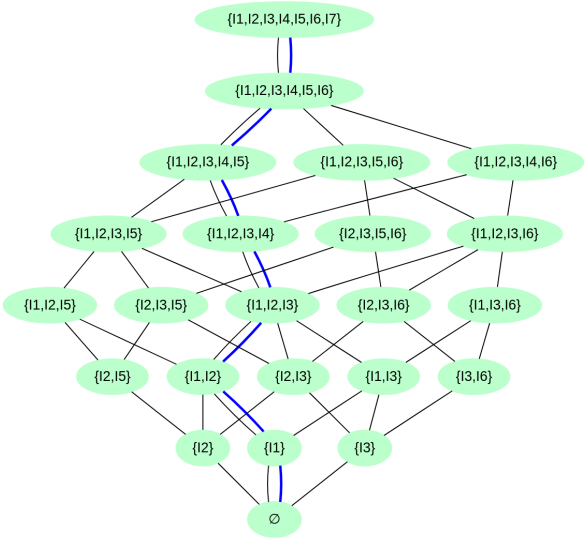

Figure 8: Learning path within a knowledge structure

the intersection of the respective surmise functions. 'kmmesh() delivers the *mesh* of two knowledge structures (Falmagne and Doignon, 1998).

`kmnotions()` computes the *notions* for a knowledge structure, i.e. the classes of equivalent items. `kmeqreduction()` gives the structure reduced to these notions. More general is `kmsubstructure()` which gives the reduction of a knowledge structure to an arbitrary subset of items. `kmexpand()` is the counterpart to that as it expands a knowledge structure to a larger items set.

Finally, there are functions providing trivial knowledge spaces: `kmminimalspace()` produces a knowledge space containing just the empty set $\emptyset$ and the full item set Q. `kmmaximalspace()`, on the other side, produces the power set 2^Q as knowledge space.

3.2.4 Working with data

There exist several functions for working with data. They can be summarized into three groups.

1. **Generating structures from data:** The function `kmgenerate()` provides a simplistic method to generate a knowledge structure from data by taking all those response patterns as knowledge states which occur with a frequency beyond a certain threshold. This method requires very high numbers of response patterns.

```
simdata <- kmsimulate(ksp, 2000, beta=0.1, eta=0.1)
kgs <- kmgenerate(simdata)
cat(str_wrap(kmprettyprint(kmbasis(kgs)), width=60))
#> {{I1}, {I2}, {I3}, {I2, I5}, {I1, I6}, {I2, I6}, {I3, I6},
#> {I1, I3, I4}, {I3, I5, I6}, {I2, I3, I4, I6}, {I1, I2, I3,
#> I4, I6, I7}}
```

   A better approach is given by the `DAKS` package described in Section 5 below. The function `kmiita2SR()` generates a surmise relation from an `iita` object obtained with `DAKS` functions. Please note, howevr, that `DAKS` produces a surmise relation

and thus a (larger) quasi-ordinal knowledge space.

2. **Validating structures with data:** There are two functions implementing different principles of validation: `kmvalidate()` delivers two distance measures based on the distribution of distances of the response patterns from the structure. `kmSRvalidate()` validates data against a surmise relation delivering two measures based on how often pairs in the relation are violated by the data.

3. **Generating data through simulation:** `kmsimulate()` simulates response patterns applying the BLIM model.

3.2.5 Knowledge assessment

Adaptive knowledge assessment was the original aim behind developing KST (Doignon and Falmagne, 1985). There is quite some literature on this topic (see, e.g., Degreef et al., 1986; Falmagne and Doignon, 1988b; Dowling and Hockemeyer, 2001; Hockemeyer, 2002; Augustin et al., 2015). An assessment routine usually has three key components:

1. A *questioning rule* selecting the next test items to be asked,
2. An *update rule* determining how to update the user's profile after evaluating their response to the test item, and
3. A *stopping criterion* to decide when enough information has been collected and the assessment is finished.

`kstMatrix` implements the stochastic procedures introduced by Falmagne and Doignon (1988b) in the notation used by Doignon and Falmagne (1999) (Chapter 10).

For the questioning rule, the *halfsplit* rule and the *informative* rule are implemented. For the update rule, the *Bayesian* and the *informative* rules are implemented. As a stopping criterion, a probability threshold can be specified; the assessment stops when the maximal likelihood of a knowledge state exceeds this threshold.

There is a general `kmassess()` function and a simpler `kmsassess()` version; the former provides for item-specific update parameters while the latter assumes that the update parameters are identical for all items. Furthermore, `kmsassess()`starts with an equal probability distribution over the knowledge structure while `kmassess()` provides for more detailed information also here.

We do an assessment with halfsplit questioning rule and Bayesian update rule assuming a general caress error probability of 0.1 and a lucky guess probability of 0.05 stopping at a probability threshold of 0.6.

```
kmsassess(c(1,1,1,1,0,0,0), ksp, "halfsplit", "Bayesian",
          0.1, 0.05, NULL, NULL, 0.6)
#> $state
#> [1] 1 1 1 1 0 0 0
#>
#> $probs
#>  [1] 1.246134e-04 2.243041e-03 1.246134e-04 2.243041e-03
#>  [5] 2.243041e-03 4.037474e-02 2.243041e-03 4.037474e-02
#>  [9] 7.267453e-01 1.311720e-05 2.361096e-04 2.361096e-04
#> [13] 4.249972e-03 7.649950e-02 2.361096e-04 4.249972e-03
```

```
#> [17] 2.361096e-04 4.249972e-03 7.649950e-02 2.485364e-05
#> [21] 4.473655e-04 8.052579e-03 8.052579e-03
#>
#> $queried
#> [1] 1 6 5 3 4
#>
#> $qtime
#> [1] 4e-04
#>
#> $utime
#> [1] 0
```

Five questions (out of 7) have been asked. This is not too much saving but the ratio improves normally considerably with higher item and state numbers. The ninth probability value with 0.727 denotes the highest one.

With an optional additional parameter `probdev=TRUE` one gets additional information on how the probability distribution has developed over the assessment process. If additionally a `directory` is specified, this is also illustrated graphically.

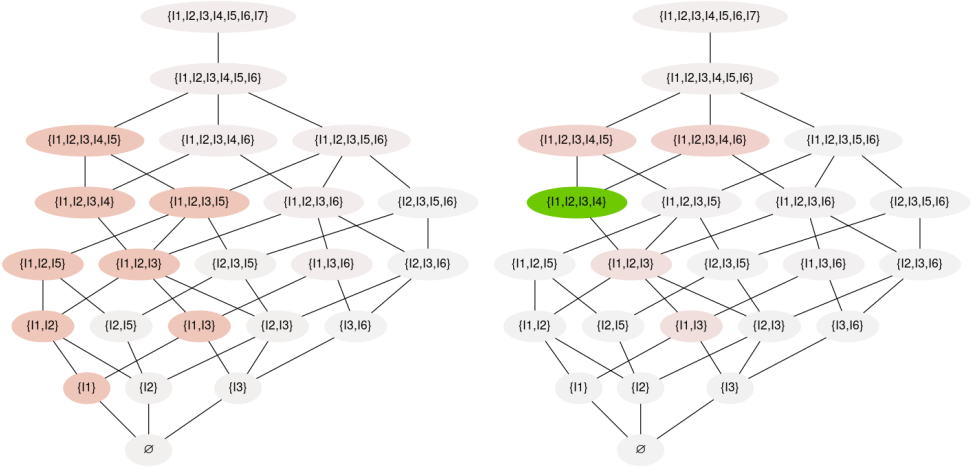

Figure 9: Assessment process steps

Figure 9 shows these illustration after the second and after the final, fifth step of the assessment. The differently colored vertices in the graph represent the different likelihoods.

3.2.6 Data sets

`kstMatrix` provides several data sets, i.e. knowledge spaces. On the one side, there are empirical structures obtained by the group around Cornelia Dowling in the early 90s (Dowling, 1993; Baumunk and Dowling, 1997) through querying experts.

- `readwrite`: Knowledge structures on reading and writing abilities
- `cad`: Abilities on working with AutoCAD
- `fractions`: Abilities on fractions

Each of these data sets is a list containing different bases.

On the other side, there are two fictional data sets, each a list with different aspects. `xpl` is a four items structure used four testing purposes in the development of `kstMatrix`, `phsg` is a data set used in a publication by Steiner et al. (ress).

3.3 `kstIO`

The `kstIO` package offers functions for reading and writing KST files. It support both, set and matrix representations. There are separate reading and writing functions for the various data types.

The reading functions return a list with two elements, `matrix` and `sets` if the respective data type is available in the `kst` package. Otherwise it is only the respective `kstMatrix` structure.

The classical KST file formats are basically binary text matrices with optional header lines providing additional information. More recently, `kstIO` has been extended to also support spreadsheet file formats, namely CSV (comma-separated values), ODS (LibreOffice/OpenOffice)„ and XLSX (Microsoft Excel).

As an example, we save our basis to an ODS file, re-read it, and look at it within OpenOffice.

```
write_kbase(phsg$basis,paste0(tempdir(), "/phsg_basis.ods"))
read_kbase(paste0(tempdir(), "/phsg_basis.ods"))
#> $matrix
#>      I1 I2 I3 I4 I5 I6 I7
#> [1,]  1  0  0  0  0  0  0
#> [2,]  0  1  0  0  0  0  0
#> [3,]  0  0  1  0  0  0  0
#> [4,]  1  1  1  1  0  0  0
#> [5,]  0  1  0  0  1  0  0
#> [6,]  0  0  1  0  0  1  0
#> [7,]  1  1  1  1  1  1  1
#> attr(,"class")
#> [1] "kmbasis"  "kmfamset" "matrix"   "array"
#>
#> $sets
#> {{"I1"}, {"I2"}, {"I3"}, {"I2", "I5"}, {"I3", "I6"},
#>  {"I1", "I2", "I3", "I4"}, <<set(7)>>}
```

4 KST extensions

4.1 `CbKST`

In CbKST, we distinguish between the level of observable response behavior to test items and the level of underlying (non-observable) skills and cimpetencies. Within each of these levels, the functions provided by the `kstMatrix` package (see above) can be used. The `CbKST` (Hockemeyer, 2026a) package caters (only) for the connection between these levels.

Core concepts in CbKST are *skill maps* and *skill multi maps.* For the former, there

	A	B	C	D	E	F	G
1	I1	I2	I3	I4	I5	I6	I7
2	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	0	1	0	0	0	0
5	1	1	1	1	0	0	0
6	0	1	0	0	1	0	0
7	0	0	1	0	0	1	0
8	1	1	1	1	1	1	1
9							

Figure 10: OpenOffice screenshot of the saved basis file

are two interpretations, the *conjunctive model* and the *disjunctive model*. Currently, the `CbKST` package supports only the conjunctive model. For this case, skill maps can be seen (and are treated by the software) as a special case of skill multi maps, i.e. as skill multi maps with exactly one competence per skill.

The starting point for working in CbKST is to read a skill (multi) map from file. This is done with `read_skillmultimap()`. ODS (OpenOffice/LibreOffice) and XLSX (Microsoft Excel) spreadsheet files are accepted.

With `cbkst_perf2comp()`, the competence state underlying an observed performance state is determined. `cbkst_competencestructure()` delivers the set of all competence states underlying some performance state. Specifically for skill maps, there is also a function `cbkst_simple_perf2comp()`.

In the other direction, there are corresponding functions `cbkst_comp2perf()` and `cbkst_performancestructure()`.

Finally, `cbkst_problemfunction()` obtains the *problem function* corresponding to a skill (mult) map.

4.2 pks

`pks` (Wickelmaier et al., 2024; Heller and Wickelmaier, 2013; Brancaccio et al., 2024) offers functions for fitting and testing knowledge structures including parameters for the BLIM and the *stochastic learnig model* (SLM).

As a preparation, we provide a set of simulated response patterns for our example structure in the NR format used by the `pks` functions.

```
simdata <- kmsimulate(ksp, 200, beta=0.1, eta=0.1)
nr <- as.pattern(simdata, freq=TRUE)
head(nr)
#> 0000000 0000001 0000010 0001100 0010000 0010010
#>       9       1       1       1       6      11
```

Applying the functions `blim()` and `slm()`, we can then estimate the respective models

```
bm <- blim(ksp, nr)
bm
#>
#> Basic local independence models (BLIMs)
#>
#> Number of knowledge states: 23
#> Number of response patterns: 60
#> Number of respondents: 200
#>
#> Method: Minimum discrepancy
#> Number of iterations: 1
#> Goodness of fit (2 log likelihood ratio):
#>  G2(91) = 104.29, p = 0.16114
#>
#> Minimum discrepancy distribution (mean = 0.285)
#>   0   1   2
#> 150  43   7
#>
#> Mean number of errors (total = 0.285)
#> careless error    lucky guess
#>      0.1150004      0.1700009
#>
#> Error and guessing parameters
#>        beta      eta
#> I1 0.037657 0.000001
#> I2 0.050885 0.000001
#> I3 0.064764 0.000001
#> I4 0.012097 0.057773
#> I5 0.000001 0.023256
#> I6 0.005780 0.035242
#> I7 0.000001 0.093264
kb <- kmbasis(bm$K)
kmprettyprint(kb)
#> [1] "{{I1}, {I2}, {I3}, {I2, I5}, {I3, I6}, {I1, I2, I3, I4}, {I1, I2, I3, I4, I
kmprettyprint(phsg$basis)
#> [1] "{{I1}, {I2}, {I3}, {I2, I5}, {I3, I6}, {I1, I2, I3, I4}, {I1, I2, I3, I4, I
```

We have exemplarily printed the basis of the resulting knowledge structure together with the original basis as comparison, and it shows identical bases. The β and η parameters, however, are partially estimated considerably lower than what we specified in the simulation (0.1 each). This is due to the fact that, in the simulation, by adding noise to some knowledge state, we may well land in some other state.

```
sm <- slm(ksp, nr)
sm
#>
#> Simple learning models (SLMs)
#>
```

```
#> Number of knowledge states: 23
#> Number of response patterns: 60
#> Number of respondents: 200
#>
#> Method: Minimum discrepancy
#> Number of iterations: 1
#> Goodness of fit (2 log likelihood ratio):
#>  G2(106) = 137, p = 0.022977
#>
#> Minimum discrepancy distribution (mean = 0.285)
#>   0   1   2
#> 150  43   7
#>
#> Mean number of errors (total = 0.28849)
#> careless error    lucky guess
#>      0.1145144      0.1739798
#>
#> Error, guessing, and solvability parameters
#>        beta      eta       g
#> I1 0.037657 0.000001 0.59750
#> I2 0.050885 0.000001 0.75333
#> I3 0.064764 0.000001 0.75917
#> I4 0.012097 0.057773 0.48723
#> I5 0.000001 0.023256 0.51881
#> I6 0.005780 0.035242 0.56970
#> I7 0.000001 0.093264 0.35000
kmprettyprint(kmbasis(sm$K))
#> [1] "{{I1}, {I2}, {I3}, {I2, I5}, {I3, I6}, {I1, I2, I3, I4}, {I1, I2, I3, I4, I
```

Here the comments on the BLIM results hold similarly.

4.2.1 Data provided by pks

pks provides several data sets

- **chess:** Data on solving chess problems by Schrepp et al. (1999)
- **DoignonFalmagne7:** A ficticious data set by Doignon and Falmagne (1999)
- **endm:** Data from Heller and Wickelmaier (2013)
- **schoolarithm:** Data on arithmetic problems for elementary and middle school (de Chiusole and Stefanutti, 2013; Stefanutti and de Chiusole, 2017)
- **Taagepera:** Data and knowledge structures on specific scice problems (Taagepera et al., 1997)

4.2.2 pksCpp - Probabilistic knowledge structures: additional models and functions

The pksCpp package is the only package presented in this paper that is not available through CRAN. It has to be installed from Github, e.g. by

```
pak::pak("jujupsy/pksCpp")
```

Primarily, `pksCpp` (Mollenhauer, 2025) re-implements functions from `pks` in C++. This holds for the conversion functions `as.binmat()` and `as.pattern()` as well as for the new function `blimCpp()`.

Furthermore, there is a new functionality by the `glimCpp()` function which provides a *generalized* local independence model.

5 Specific functionalities

5.1 CDSS

`CDSS` (Hockemeyer, 2026b) aims at deriving structures from an assignment of taught and required, respectively, skills to learning objects in an existing course. These structures are to be seen as rapid prototypes which need further refinement. Nevertheless, it should drastically simplify the process of developing such structures.

As first step, a skill assignment is read from a respective spreadsheet file using `cdss_read_skill_assignment()`. This function checks certain conditions and rejects the data by default if these conditions are not met. Afterwards, attribution functions on skill and learning object level can be obtained through `cdss_skill_sa2af()`and `cdss_lo_sa2af()`, respectively.

Under certain conditions, a skill assignment may also lead to attribution relations instead of attibution functions. This can be checked with `cdss_sa_describes_sr()`. If `TRUE`, the attribution relations can be determined with `cdss_skill_sa2ar()` and `cdss_lo_sa2ar()`.

Attribution functions and attribution relations can subsequently be closed to surmise functions and surmise relations, respectively, applying `kstMatrix::kmclosure(). These can then be further processed using`kstMatrix‘ (see above) functions.

Furthermore, `CDSS` offers the `cdss_sa2mu()` function which creates an interface to the `CbKST` package (see above)

5.2 DAKS

The `DAKS` package (Sargin and Ünlü, 2016; Ünlü and Sargin, 2010) provides functions and a data set for the *inductive item tree analysis*. Most of its functions are primarily aimed for internal usage. The main function is `iita()`, its second parameter selects the concrete method to be used.

```
simdata <- kmsimulate(ksp, 100, beta=0.1, eta=0.1)
i <- iita(simdata, v=1)
i
#>
#>       Inductive Item Tree Analysis
#>
#> Algorithm: minimized corrected IITA
```

```
#>
#> quasi order: {(1L, 7L), (2L, 4L), (2L, 5L), (2L, 7L), (3L, 4L),
#>  (3L, 7L), (6L, 7L)}
```

```
hasse(i$implications, items=7)
```

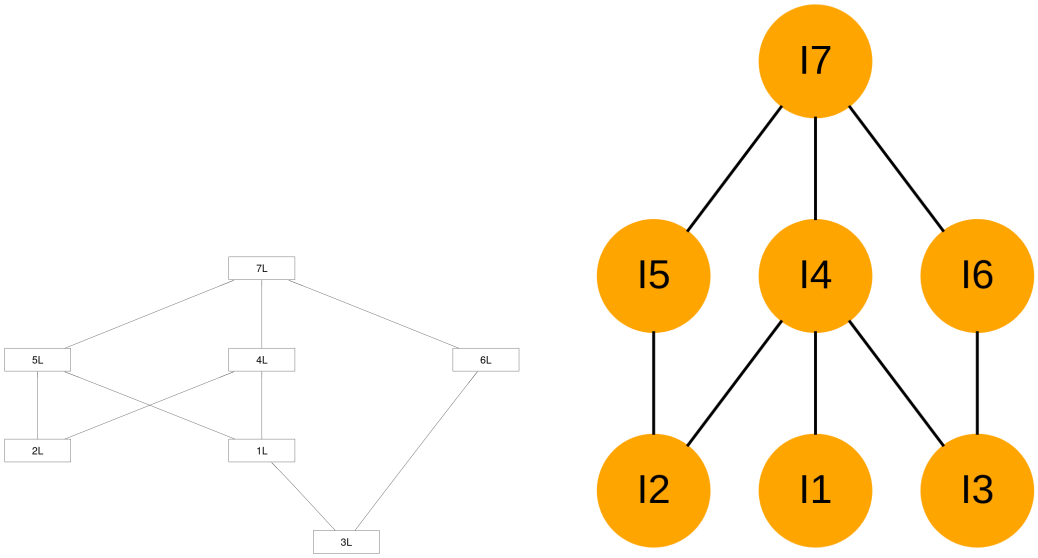

Figure 11: Implications determined by 'DAKS' compared to the true relation

5.2.1 Data provided by `DAKS`

`DAKS` provides a data set from the 2003 PISA (Programme for International Student Assessment) test. It contains 340 response patterns on a 5-item-test on mathematical literacy.

5.3 `mycaas`

`mycaas` (Brancaccio and Granziol, 2024) provides functions and Shiny apps for the adaptive assessment of knowledge. Actually, the *halfsplit* questioning rule and the *multiplicativ* update rule are applkied.

The core routine is the `assessment()` function. Several parts of the assessment are realized in own functions. `mycaas` includes two Shiny app functions `run_Assessment()` and `run_Practice()`. The result can then be further inspected, e.g. plotting a Hasse diagram of the knowledge structure with the assessed knowledge state and its neighbors being marked.

5.3.1 Data

The data set `AA_knowledge_test` serves as an example for testing the functions of `mycaas`. It contains a six-items-test on computerized adaptive assessment.

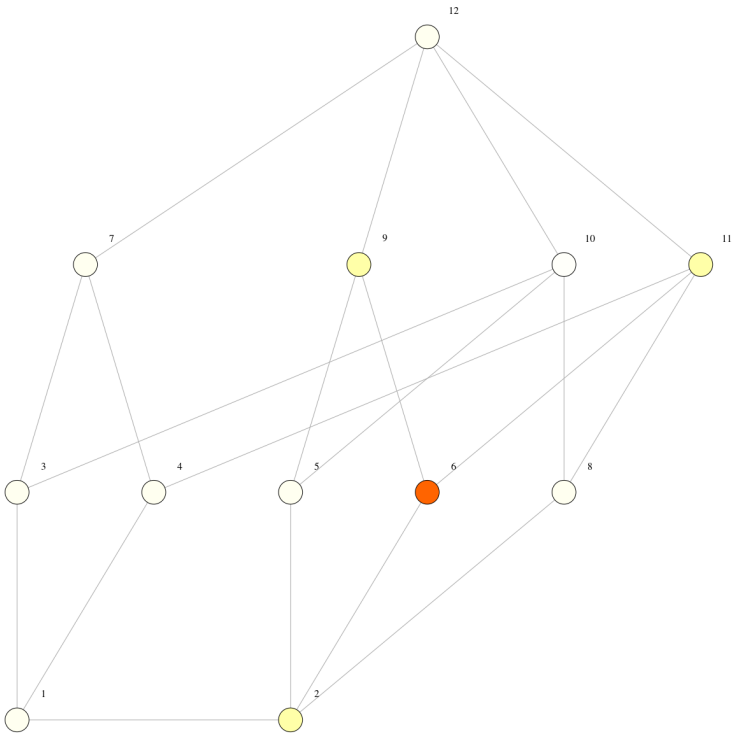

Figure 12: “mycaas‘ assessment result

List of described R packages

- CbKST
- CDSS
- DAKS
- kst
- kstIO
- kstMatrix
- mycaas
- pks
- pksCpp

References

Albert, D. and Lukas, J., editors (1999). *Knowledge Spaces: Theories, Empirical Research, Applications*. Lawrence Erlbaum Associates, Mahwah, NJ.

Augustin, T., Hockemeyer, C., Suck, R., Podbregar, P., Kickmeier-Rust, M. D., and Albert, D. (2015). Individualized skill assessment in educational games: The mathematical foundations of partitioning. *Journal of Mathematical Psychology*, 67:1–7.

Baumunk, K. and Dowling, C. E. (1997). Validity of spaces for assessing knowledge about fractions. *Journal of Mathematical Psychology*, 41:99–105.

Brancaccio, A., de Chiusole, D., and Wickelmaier, F. (2024). Software packages for knowledge space theory. In Heller, J. and Stefanutti, L., editors, *Knowledge Struc-*

tures: Recent Developments in Theory and Application, volume 7 of *Advanced Series on Mathematical Psychology*, pages 297–322. World Scientific, Singapore.

Brancaccio, A. and Granziol, U. (2024). *mycaas: My Computerized Adaptive Assessment*. University of Padua, Italy. R package version 0.0.1.

de Chiusole, D. and Stefanutti, L. (2013). Modeling skill dependence in probabilistic competence structures. *Electronic Notes in Discrete Mathematics*, 42:41–48.

Degreef, E., Doignon, J.-P., Ducamp, A., and Falmagne, J.-C. (1986). Languages for the assessment of knowledge. *Journal of Mathematical Psychology*, 30:243–256.

Doignon, J.-P. (1994). Knowledge spaces and skill assignments. In Fischer, G. H. and Laming, D., editors, *Contributions to Mathematical Psychology, Psychometrics, and Methodology*, pages 111–121. Springer–Verlag, New York.

Doignon, J.-P. and Falmagne, J.-C. (1985). Spaces for the assessment of knowledge. *International Journal of Man-Machine Studies*, 23:175–196.

Doignon, J.-P. and Falmagne, J.-C. (1999). *Knowledge Spaces*. Springer–Verlag, Berlin.

Dowling, C. E. (1993). Applying the basis of a knowledge space for controlling the questioning of an expert. *Journal of Mathematical Psychology*, 37:21–48.

Dowling, C. E. and Hockemeyer, C. (2001). Automata for the assessment of knowledge. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):451–461.

Düntsch, I. and Gediga, G. (1995). Skills and knowledge structures. *British Journal of Mathematical and Statistical Psychology*, 48:9–27.

Falmagne, J.-C. and Doignon, J.-P. (1988a). A class of stochastic procedures for the assessment of knowledge. *British Journal of Mathematical and Statistical Psychology*, 41:1–23.

Falmagne, J.-C. and Doignon, J.-P. (1988b). A Markovian procedure for assessing the state of a system. *Journal of Mathematical Psychology*, 32:232–258.

Falmagne, J.-C. and Doignon, J.-P. (1998). Meshing knowledge structures. In Dowling, C. E., Roberts, F. S., and Theuns, P., editors, *Recent Progress in Mathematical Psychology*, Scientific Psychology Series, pages 143–153. Lawrence Erlbaum Associates Ltd., Mahwah, NJ.

Heller, J. and Stefanutti, L., editors (2024). *Knowledge Structures: Recent Developments in Theory and Application*, volume 7 of *Advanced Series on Mathematical Psychology*. World Scientific, Singapore.

Heller, J. and Wickelmaier, F. (2013). Minimum discrepancy estimation in probabilistic knowledge structures. *Electronic Notes in Discrete Mathematics*, 42:49–56.

Hockemeyer, C. (2002). A comparison of non–deterministic procedures for the adaptive assessment of knowledge. *Psychologische Beiträge*, 44:495–503.

Hockemeyer, C. (2026a). *CbKST: Functions for Competence–based Knowledge Space Theory*. Department of Psychology, University of Graz, Austria. R package version 0.1-1.

Hockemeyer, C. (2026b). *CDSS: Course–Dependent Skill Structures*. Department of Psychology, University of Graz, Austria. R package version 0.3-1.

Hockemeyer, C., Steiner, P., and Wong, W. (2026). *kstMatrix: Basic Functions in Knowledge Space Theory Using Matrix Representations*. Department of Psychology, University of Graz, Austria. R package version 3.0-1.

Korossy, K. (1997). Extending the theory of knowledge spaces: A competence–performance approach. *Zeitschrift für Psychologie*, 205:53–82.

Mollenhauer, J. (2025). *pksCpp: Probabilistic Knowledge Structures — Additional Models and Functions*. University of Tübingen, Germany.

Sargin, A. and Ünlü, A. (2016). *DAKS: Data Analysis and Knowledge Spaces*.

Schrepp, M., Held, T., and Albert, D. (1999). Component–based construction of surmise relations for chess problems. In Albert, D. and Lukas, J., editors, *Knowledge Spaces: Theories, Empirical Research, Applications*, pages 41–66. Lawrence Erlbaum Associates, Mahwah, NJ.

Stahl, C. (2008). *Developing a Framework for Competence Assessment*. Unpublished dissertation, Vienna University of Economics and Business.

Stahl, C., Meyer, D., and Hockemeyer, C. (2026). *kst: Knowledge Space Theory*. Department of Psychology, University of Graz, Austria. R package version 0.5-5.

Stefanutti, L. and de Chiusole, D. (2017). On the assessment of learning in competence based knowledge space theory. *Journal of Mathematical Psychology*, 80:22–32.

Steiner, P., Hockemeyer, C., Kickmeier-Rust, M., and Hochweber, J. (in press). Structuring and assessing knowledge with knowledge space theory. In Wilson, M. L., Tawfik, A. A., Ritzhaupt, A. D., and Lowdermilk, A. T., editors, *Quantitative Methods in Educational Research:: Concepts and Applications*. EdTech Books.

Taagepera, M., Potter, F., Miller, G. E., and Lakshminarayan, K. (1997). Mapping students' thinking patterns by the use of knowledge space theory. *International Journal of Science Education*, 19:283–302.

Ünlü, A. and Sargin, A. (2010). DAKS: An R package for data analysis methods in knowledge space theory. *Journal of Statistical Software*, 37:1–31.

Wickelmaier, F., Heller, J., Mollenhauer, J., and Anselmi, P. (2024). *pks: Probabilistic Knowledge Structures*. University of Tübingen, Germany.