

Project *Surmise Relations between Tests**

KST Tools User Manual

2nd edition

Cord Hockemeyer

January 26, 2001

Institutsbericht Nr. 2001/3
Institut für Psychologie
Karl–Franzens–Universität Graz, Austria

*This project is financed by the Austrian National Science Fund (FWF) under the project number P12726-SOZ granted to Dietrich Albert and Wilhelm Schappacher.

©2001, Cord Hockemeyer
Abteilung für Allgemeine Psychologie
Institut für Psychologie
Karl-Franzens-Universität Graz, Austria

All rights reserved.

An earlier version was developed by Cord Hockemeyer at the University of Technology at Braunschweig, Germany.

This text was produced with L^AT_EX 2_ε. A PostScript version was produced with `dvips` and a PDF version with PDFL^AT_EX.

T_EX is a trademark of the American Mathematical Society (AMS); PostScript and PDF are trademarks of Adobe, Inc.; UNIX is a registered trademark licensed by the X/Open Company, Ltd.; ANSI is a registered trademark of the American National Standards Institute; SUN, Solaris, and Java are trademarks of SUN Microsystems Inc. All other product names mentioned herein are trademarks of their respective owners.

Contents

1	Overview	1
1.1	Some Introductory Remarks	1
1.2	Notations in this documentation	1
1.3	Program Overview	2
2	KST Specific Tools	7
2.1	basis	7
2.2	basis-union	8
2.3	constr	8
2.4	dep-pairs	10
2.5	distance	11
2.6	heights	15
2.7	mk-empty-space	16
2.8	rule-error-table	16
2.9	select	18
2.10	sigma-distance	19
3	General Tools	21
3.1	count	21
3.2	permute	21

Contents

4	File Formats	23
4.1	basisfile	23
4.2	patternfile	24
4.3	spacefile	25

1 Overview

1.1 Some Introductory Remarks

This manual describes a set of programs which have been written by Cord Hockemeyer to facilitate work with knowledge spaces. It updates an earlier version (Hockemeyer, Cord & Dowling, Cornelia E. (1996): *KST Tools User Manual*. Internal report, Department of Psychology, University of Technology, Braunschweig, Germany) by adding some new programs and updating others, existing ones.

Actually, this document is a compilation consisting of a general overview document (Chapter 1) and a set of man pages for a selected subset of programs (Chapter 2 – 4). This may lead to a slightly incoherent style within the manual.

1.2 Notations in this documentation

Files contain in general a data matrix preceded by two header lines specifying the number of columns and the number of rows. Knowledge space files¹ and answer pattern files² have matrices of 0's and 1's; basis files have matrices of 0's, 1's, and 2's. In knowledge space files³, a '1' means that the item described by the column is element of the state described by the row. In answer pattern files⁴, a '1' describes a student (row) who has answered a question for an item (column) correctly. In both file types we use a '0' otherwise. In basis files we use a '1' to denote that a basis element (row) is minimal for an item (column), a '2' to show that the basis element just contains the item and a '0' otherwise.

For each program there is a note stating the program language and - if appropriate - other programs used. There are three languages used: *C*, *C++*, and *bash* (the latter meaning shell scripts using the bash). The scripts may be not portable. For the shell

¹<http://wundt.kfunigraz.ac.at/hockemeyer/spacefile.html>

²<http://wundt.kfunigraz.ac.at/hockemeyer/patternfile.html>

³<http://wundt.kfunigraz.ac.at/hockemeyer/spacefile.html>

⁴<http://wundt.kfunigraz.ac.at/hockemeyer/patternfile.html>

1 Overview

scripts the *C/C++* programs used within which have been written especially for knowledge spaces are also mentioned. Other programs used in the scripts, however, which are part of the UNIX operating system and its standard utilities have not been mentioned. Examples for the latter are *sed*, *cut*, *paste*, *[ef]grep*, *head*, or *tail*.

Most of these programs should be available via ftp for members of cooperating groups until July 1996. As far as they are available, their documentation will also be available via WWW. The documentation exists in several formats, HTML, man-page, TeX, and Postscript. Links to already existing parts of the documentation are placed in this text.

1.3 Program Overview

Different Representations

- **basis:**⁵ Computes the basis of a knowledge structure. (C)
- **constr:**⁶ Constructs the knowledge space for a given knowledge structure. The original structure must be in ASCII space file format or in basis file format. The format of the space file (ASCII or binary) can be chosen via command line options. (C)

Different File Formats

- **a2b-space:** Convert a spacefile or patternfile from ASCII into binary format. (C)
- **b2a-space:** Convert a spacefile or patternfile from binary into ASCII format. (C)

Working with Rules

Comment: Rules is used here in the sense of surmise systems.

- **rule-cnt:** Just count the rules rejected by a basis. (C, *needs a lot of computing time if you have items with many clauses*)

Informations

- **basis-print:** Print a list of the basis elements as number lists with one element per line. Items for which an element is minimal are marked with an underscore. (C++, *it is a good idea to postprocess the list for including it into e.g. TeX-sources*)

⁵<http://wundt.kfunigraz.ac.at/hockemeyer/basis.html>

⁶<http://wundt.kfunigraz.ac.at/hockemeyer/constr.html>

- **equivalence:** Compute and print classes of equivalent items in a knowledge spaces given by its basis. (C)
- **heights:**⁷ Compute for each item its height within a knowledge space defined as the minimal size of its clauses. (C)

Combining Different Knowledge Spaces

- **basis-union:**⁸ Compute the basis of the union of two knowledge spaces given by their bases. (*bash, using the basis program mentioned above*)
- **basis-section:** Computes the basis of the section of two knowledge spaces described by their bases. (C; *still has limited number of items and basis elements which are set at compiling time*)

Comparing Different Knowledge Spaces

- **comp-rules:** Compute different coefficients concerning rules common to two knowledge spaces. (*bash, using the rule-cnt program mentioned above*)
- **sigma-distance:**⁹ Compute for each item coefficients concerning their clause sets in the surmise systems. (C)

Changing Knowledge Spaces

- **s-closure:** Compute the closure under intersection of a knowledge structure. The resulting structure is printed to `|samp;stdout|/samp;.` (*bash, using the constr program mentioned above*)
- **su-closure:** Compute the closure under intersection of a knowledge structure. The basis of the resulting structure is printed to `|samp;stdout|/samp;.` (*bash, using the basis and constr programs mentioned above*)
- **red-basis:** Reduce a basis to a subset of items. (*bash, using the basis program mentioned above*)
- **select-cols:** Extension of the *red-basis* program. Order of items in the destination basis can be freely selected. Does not allow regions in the specification of the item list. (*bash, using the basis program mentioned above*)

⁷<http://wundt.kfunigraz.ac.at/hockemeyer/heights.html>

⁸<http://wundt.kfunigraz.ac.at/hockemeyer/basis-union.html>

⁹<http://wundt.kfunigraz.ac.at/hockemeyer/sigma-distance.html>

1 Overview

- **permute-basis:** Permute the columns of a basis according to a random permutation. (*bash, using a small C-program called permute for randomly choosing a permutation*)
- **select:**¹⁰ Select randomly a number of knowledge states from a given knowledge space and store it in the format of a knowledge space. (*C*)

Creating special Knowledge Spaces

- **diagonal-basis:** Create a basis building a diagonal matrix, i.e. a basis representing the knowledge space which contains all subsets of the item set. (*bash, using a C-program called count; should be rewritten in C for efficiency*)
- **mk-empty-space:**¹¹ Just the complement program to *diagonal-basis*. Write a spacefile which contains only the empty set and the complete item set as its knowledge states. (*C*)
- **random-patterns:** Create a family of randomly selected subsets in binary patternfile/spacefile format. (*C*)

Working with Surmise Relations

- **dep-matrix:** Show a surmise relation as a matrix. (*C, link to dep-pairs*)
- **dep-pairs:**¹² Show a surmise relation as a list of pairs. (*C, behaving depending on its name*)
- **dep-all:** Show for each pair of items whether or not it is contained in the surmise relation given. (*C, link to dep-pairs*)
- **comp-s-closure-rules:** Compute different coefficients concerning rules common to two knowledge spaces which must be closed under intersection. (*bash, using the base-union, dep-pairs, and s-closure programs mentioned above*)

Validation Using Student Data

- **rule-errors:** Compute for each pair in a surmise relation the number of students' answer patterns which contradict or support the pair. The resulting table has six rows: the items in the pair, and the numbers of contradicting, corresponding, (1,1), and (0,0) pairs. (*bash, using the dep-pairs program mentioned above*)

¹⁰<http://wundt.kfunigraz.ac.at/hockemeyer/select.html>

¹¹<http://wundt.kfunigraz.ac.at/hockemeyer/mk-empty-space.html>

¹²<http://wundt.kfunigraz.ac.at/hockemeyer/dep-pairs.html>

- **all-rule-errors:** Create for a given basis and a set of answer patterns a table of all pairs of different items describing whether or not the expert has accepted the pair as member of the surmise relation and how many answer patterns contradict this pair. (*bash, using the dep-all program mentioned above*)
- **rule-error-table:**¹³ Print a table which shows for each pair of different items how many answer patterns in a datafile contradict this pair and, for a list of bases representing surmise relations, whether or not the pair is an element of the surmise relations.
- **quad-table:** Print for a set of answer patterns and for all 2-item-sets a table showing the frequencies of the four possible 0-1-pairs of managing the items. Each table is printed into an own file. (*bash, using just a small utility called count which prints the list of numbers from 1 to n*)
- **distance:**¹⁴ Compute the frequency distribution for the different possible distances between students' answer patterns and the states of a knowledge space given in binary format. This program was strongly influenced by a program named *di.exe* which was originally written by Theo Held¹⁵. In the meantime, it has been extended to the computation of several measures on the invalidity of items and prerequisite relationships. (*C*)
- **clause-val:** This program computes a measure for the validity of item-clause pairs in a given basis with respect to a specified data set of answer patterns. (*C*)

Simulating Students

- **simple-sim:** Create a set of virtual answer patterns based on a given knowledge space by simulating students using item-independent β and *eta* values for probabilities of careless errors and lucky guesses, respectively.
- **elab-sim:** Create a set of virtual answer patterns based on a given knowledge space by simulating students using item-dependent β and *eta* values for probabilities of careless errors and lucky guesses, respectively, which are retrieved through a parameter file.

¹³<http://wundt.kfunigraz.ac.at/hockemeyer/rule-error-table.html>

¹⁴<http://wundt.kfunigraz.ac.at/hockemeyer/distance.html>

¹⁵<mailto:T.Held@psych.uni-halle.de>

1 Overview

2 KST Specific Tools

2.1 basis — Computing the basis of a knowledge structure

Last changed: 4 February 1999

Synopsis

```
basis [options] structurefile basisfile
```

Description

`basis` computes the basis of a knowledge structure. The structure may be stored in a `spacefile` (Sect. 4.3, p. 25) in either binary or ASCII format. The resulting basis is stored in `basisfile` (Sect. 4.1, p. 23) format. The elements of the basis are ordered by their size so it can be used directly with the `constr` (Sect. 2.3, p. 8) program.

Usage

```
basis [Options] structurefile basisfile
```

Using `-` for `structurefile` or `basisfile` indicates the usage of `stdin` or `stdout`, respectively, as files. This also enables usage as a filter command.

Options are:

`-a` | `--ascii`: Use ASCII format for `structurefile` (see *spacefile*, sect. 4.3).

`-b` | `--binary`: Use binary format for `structurefile` (see *spacefile*, sect. 4.3).

Only one of these options may be used.

Bugs

Currently the *optional* file format specification is mandatory.

See also

basisfile (Sect. 4.1, p. 23), spacefile (Sect. 4.3, p. 25) , constr (Sect. 2.3, p. 8)

2.2 basis-union — Computing the basis representing the union of two knowledge spaces

Last changed: 19 April 1999

Synopsis

```
basis-union basis1 basis2 union
```

Description

`basis-union` computes the basis representing the union of two given knowledge spaces. Both knowledge spaces are represented as bases (see *basisfile*, sect. 4.1).

In a first step, the original bases `basis1` and `basis2` are concatenated. In a second step, then, the minimal elements of the knowledge structure resulting from the first step are determined and stored in the file `union`. For this second step, the `basis (5K)` program is used.

See also

spacefile (Sect. 4.3, p. 25), basisfile (Sect. 4.1, p. 23) , basis (Sect. 2.1, p. 7)

2.3 constr — Computing the closure under union

Last changed: 11 July 2000

Synopsis

```
constr [options] structure space
```

Description

`constr` computes the closure under union of a family of sets. `constr` uses the algorithm developed by Cornelia Dowling (1993). Its main usage is the construction of knowledge spaces (see *spacefile*, sect. 4.3) from their bases (see *basisfile*, sect. 4.1).

Usage

```
constr [Options] structure space
```

`structure` may be in `spacefile` (Sect. 4.3, p. 25) or `basisfile` (Sect. 4.1, p. 23) format. In principle, the states can be in any order. Note, however, that ordering the knowledge states in `structure` by their size, i.e. by the numbers of elements contained in the state, enhances the efficiency of the closure procedure.

For both, `basisfile` and `spacefile`, a dash (' - ') may be used to indicate `stdin` or `stdout`, respectively, as a file.

Options are:

- a | --ascii : Use ASCII format for `spacefile` (see *spacefile*, sect. 4.3).
- b | --binary : Use binary format for `spacefile` (see *spacefile*, sect. 4.3).
- c | --complement : Compute the complements of the states before storing the space
- v | --verbose : Select informative output.

See also

`spacefile` (Sect. 4.3, p. 25), `basisfile` (Sect. 4.1, p. 23)

Cornelia Dowling (1993): On the Irredundant Construction of Knowledge Spaces. *Journal of Mathematical Psychology*, 37:49-62

2.4 **dep-pairs dep-all dep-matrix relation — Print dependence information in a surmise relation**

Last changed: 27 August 1999

Synopsis

```
dep-pairs basisfile  
dep-all basisfile  
dep-matrix basisfile  
relation basisfile
```

Description

`dep-pairs` prints information on prerequisite relationships between items in a surmise system. The surmise system is read from `basisfile`. The form of the output depends on the name the program was called with.

`dep-pairs` prints a list of all pairs (q, p) of different items where, from mastering item q , one can conclude the mastery of item p . Note, however, that only the numbers of the items are printed.

`dep-all` prints a list specifying for all pairs (q, p) of different items whether p is a prerequisite of q (printing a '1') or not (printing a '0'.) Here, also only the item numbers and the dependence flag are printed.

`dep-matrix` prints a (Q, Q) matrix with dependence information. A '1' in row q and column p indicates that p is a prerequisite for q . Otherwise, a '0' is printed. For $q=p$, a '-' is used. If the set of items has less than 40 elements, then the columns of the matrix are separated by a space, otherwise they are not separated at all.

`relation` prints a similar (Q, Q) matrix as `dep-matrix`. However, header information according to the `libsrbi` relation file format are printed first. In the matrix itself, for $q=p$, a '1' is printed instead of the '-' used by `dep-matrix`, and even for small item sets with less than 40 items, no separating spaces are printed.

See also

`basisfile` (Sect. 4.1, p. 23)

2.5 **distance di invalidity prereq — Computing the distance between two knowledge spaces**

Last changed: 11 June 1999

Synopsis

```
distance [options] datafile spacefile
di [options] datafile spacefile
invalidity [options] datafile spacefile
prereq [options] datafile spacefile
```

Description

`distance` computes the distance between two knowledge spaces or between a set of answer patterns and a knowledge space. The distance is defined as the average of the distances between the single answer patterns and the knowledge space. This latter distance is defined as the minimal size of the symmetrical set difference between answer pattern and knowledge state for all knowledge states within the knowledge space.

Depending on the selected options, all, some, or none of the distance values are printed. If all distances are printed an additional list of nearest knowledge states for those answer patterns which are not contained in the knowledge space can be requested. This list contains, one knowledge state per line, the position of the knowledge state within the spacefile and a list of differing items which are additionally marked with an 'e' or a 'g' to indicate whether the student has made a careless error or a lucky guess, respectively. The possibility to get a table of only those patterns whose distance exceeds a specified warning level offers a first step towards finding out extreme answer patterns in the validation process.

Optionally, `distance` also computes measures for the invalidities of items. These measures for invalidities have been developed by Katja Baumunk, Cornelia Dowling, and Cord Hockemeyer. Here, you find also a distinction between items which may have been luckily guessed and items where the student may have erred carelessly.

`distance` evaluates the name it was evoked with. Depending on this name, the default behaviour changes. For details, see below at the end of the section *Usage*.

Usage

distance datafile spacefile [options]
invalidity datafile spacefile [options]
prereq datafile spacefile [options]
di datafile spacefile [options]

Options:

- a | --ascii: Assume following file(s) to be in ASCII format.
- b | --binary: Assume following file(s) to be in binary format.
- c | --computer: Print prerequisite invalidities in a easily computer-readable format.
- d | --dist-only: Do not compute invalidity measures.
- i | --item: Compute the invalidity of items.
- l | --list: Print for each pattern a list of the nearest states (implies --table)
- m | --missing: Print list of states which do not occur in the data set.
- n | --neighbours: Also print weighted distribution of nearest neighbours (positions in spacefile; implies --states).
- NN | --number=N: Print at most N nearest states for each answer pattern (implies --list and --table). Default for N is 10.
- p | --prereq: Compute the invalidity of prerequisite relationships (implies --item).
May need much memory!
- q | --quiet: Do not produce informative output.
- s | --states: Print distribution of actually registered knowledge states (positions in spacefile).
- t | --table: Print a table with distances for each answer pattern.
- WW | --warning=W: Print a table with all answer patterns with a distance of at least W.
- debug: Produce debug output.

Note that options may **not** be concatenated but must be specified, i.e. you may **not** use -pl for -p -l!

Defaults:

```

distance: --binary --dist-only --quiet (general default)
di: --ascii --dist-only --table --list
invalidity: --binary --item
prereq: --binary --item --prereq

```

Both files have to be knowledge space files (see *spacefile*, sect. 4.3) or answer pattern files (see *patternfile*, sect. 4.2) in the selected format. The filenames may not start with a dash. Note, however, that the filename ' - ' itself may be specified to use `stdin` as input file.

Results

In the following, the maximum output is described. Depending on the command name and the options specified, some parts may be omitted in your output.

The first table - which actually is the part inspired by Theo Held - describes for each answer pattern the nearest knowledge states and the differences between the pattern and these states. For each pattern, we get one line containing the number of the pattern, the distance to the knowledge space, and the pattern itself (as a set of item numbers). For each of the nearest neighbours, we get a line with three columns: the number of the state in the spacefile, the number of nearest neighbours, and finally the difference between pattern and actual state. In this difference, items which are member of the pattern but not of the state are marked with a *g*, while those items which are element of the state but not of the answer pattern are marked with an *e*. Both types of rows are printed in a way such that each number column can be separated for automated postprocessing.

The second table contains the frequency distribution of the pattern-state distances. This table is preceded by the average distance and the standard deviation.

The third table contains the *missing states*, i.e. those states which have not been observed in the set of subsets. If the `--neighbours` option was specified, only those states are listed which additionally are not considered as *possibly underlying knowledge state* in the sense of being a nearest neighbour to any observed answer pattern.

The next table contains the state distribution. If only the `--states` option was specified, this table lists, for each knowledge state observed in the set of subjects, the frequency of its observation. If the `--neighbours` option was specified, it lists all states which have been nearest neighbours with the respective weighted frequency. Weighted frequency here means, for example, that if a pattern has three nearest neighbours then each of these neighbours is counted one third. If an answer pattern fits to a knowledge state exactly, then this state is regarded as the patterns (only) nearest neighbour.

2 KST Specific Tools

The tables regarding invalidities will be documented later.

In the `Error_p` matrix, the i th row and j th column contains the i th component of the `Error_p[j]` vector. The same holds for the `Guess_p` matrix.

History

The first idea for this program was taken from the program `di.exe` written by Theo Held. It was completely rewritten by Cord Hockemeyer using internal data representations needing less memory.

1995-07-18: distance Version 1.0 Only computing the distances between binarily stored knowledge spaces.

1996-05-02: Version 2.0 Computing distances and (optionally) invalidity measures for knowledge spaces and answer pattern files in ASCII or binary format.

1996-05-15: Version 2.1 Adding output of a list of distance and nearest states for each answer pattern as it was done in the original `di.exe` by Theo Held.

1996-08-28: Version 2.2 Extending item related invalidity measures for separated analysis of careless errors and lucky guesses.
Changing the list format from list of nearest states to list of postitions of nearest states in the spacefile together with list of differing items.

1997-01-25: Version 2.3 Realizing the long-promised `prereq` option.

1997-02-28: Version 2.3.1 Adding computer option to make statistical processing of the program output easier.
Adding validity option for output of validity measures instead of invalidity measures.
Eliminate a bug in the quiet option.
Correcting the `set_size()` function (ENDIAN problem).

1997-05-15: Version 2.4 Adding output of version number and date.
Adding the output of the number of each exactly met knowledge state.
Adding number of nearest neighbours for each answer pattern.
Adding optional output of weighted frequency of states `_and_` neighbours observed.
Improving the output format.

1997-07-08: Version 2.4.1 Numbering states with 1 to n instead of 0 to $n-1$.
Refinement of output of state list.

1997-08-30: Version 2.4.2 Refinement of output of state list.

1999-02-24: Version 2.4.3 Computer readable output of distance distribution.

1999-06-11: Version 2.4.4 Adding missing option.

Bugs and comments

The invalidity measures are still under development. Therefore, they are not yet documented in detail.

There exist two more programs dealing with distances between answer patterns and knowledge spaces: *sigma-distance* and *clause-val*.

See also

patternfile (Sect. 4.2, p. 24), *spacefile* (Sect. 4.3, p. 25) , *sigma-distance* (Sect. 2.10, p. 19)

2.6 heights — Compute the heights of items in a knowledge space

Last changed: 11 April 1996

Synopsis

`heights basisfile`

Description

`heights` computes the heights of items in a knowledge space represented by its basis. The height of an item is defined as the minimal size of the knowledge states containing these items.

See also

basisfile (Sect. 4.1, p. 23)

2.7 **mk-empty-space** — Create an almost empty knowledge space

Last changed: 11 April 1998

Synopsis

```
mk-empty-space [-a|--ascii|-b|--binary] itemno spacefile
```

Description

`mk-empty-space` creates an almost empty knowledge space, i.e. a knowledge space which contains only two knowledge states: the empty set and the complete item set.

Usage

```
mk-empty-space [options] itemno spacefile
```

Options are:

- a|--ascii: Use ASCII format for *spacefile* (see *spacefile*, sect. 4.3).
- b|--binary: Use binary format for *spacefile* (see *spacefile*, sect. 4.3).

Only one of these options may be used.

See also

spacefile (Sect. 4.3, p. 25)

2.8 **rule-error-table** — Compare and validate surmise relations

Last changed: 22 August 1996

Synopsis

```
rule-error-table datafile basisfile1...
```

Description

`rule-error-table` prints information for comparing and validating surmise relations. The surmise relations are read from the basisfiles. There must be at least one basisfile but their total number is not limited.

The information is given as a table with one row for each pair of different items. The first column contains the pair of items under consideration. The items are printed as three digit numbers with leading zeroes.

The following six columns describe the distribution of the subjects to the different response patterns with regard to the two items in the following order:

1. Number of subjects who answered the first item correctly
2. Number of subjects who answered both items correctly
3. Number of subjects who answered only the first item correctly
4. Number of subjects who answered the only the second item correctly
5. Number of subjects who answered both items incorrectly
6. Total number of subjects

Therefore, the first of these six columns should equal the sum of the second and third column, and the last of these six columns should equal the sum of the second to the fifth column. These numbers are also printed as three digit numbers with leading zeroes.

The following columns show for each of the pair of items, whether or not is is an element of the surmise relations given by the `basisfile` parameters. Each column contains a '1' if the pair of items is an element of the corresponding surmise system and a '0', otherwise.

Internally, this script uses the `dep-all` (see *dep-pairs*, sect. 2.4) program.

History

This script was written by Cord Hockemeyer (Cord.Hockemeyer@kfunigraz.ac.at) on July 22, 1996, based on discussions with Dietrich Albert (Dietrich.Albert@kfunigraz.ac.at) and Ernst Taeubl (Ernst.Taeubl@kfunigraz.ac.at).

2 *KST Specific Tools*

It was extended on August 22, 1996, after first usage had shown the need for more accurate information about the distribution of response patterns.

See also

dep-pairs (Sect. 2.4, p. 10), basisfile (Sect. 4.1, p. 23) , patternfile (Sect. 4.2, p. 24)

2.9 select — Select randomly a set of knowledge states from a knowledge space

Last changed: 11 April 2000

Synopsis

```
select no_of_states spacefile selectionfile [seed]
```

Description

`select` selects randomly `no_of_states` knowledge states from the knowledge space stored in `spacefile` and stores these states in `selectionfile`. Both files are in binary `spacefile` (see *spacefile*, sect. 4.3) format. The optional parameter `seed` gives an initialization value for the `srand(3)` function. Otherwise the result of the `time(2)` function is used.

ToDo

This program should be extended in order to enable any combinations of ASCII and binary files for the complete knowledge space file and the file of selected states. `spacefiles` (see *spacefile*, sect. 4.3).

See also

`spacefile` (Sect. 4.3, p. 25), `srand (3)` , `time (2)`

2.10 **sigma-distance** — Computing the distance between two knowledge spaces

Last changed: 10 April 1996

Synopsis

```
sigma-distance basisfile1 basisfile2 [-v|--verbose]
```

Description

`sigma-distance` computes various measures for the distance between two knowledge spaces based on a per item comparison of their surmise systems. The exact definition of these distance measures is printed if `select-distance` is called with the `verbose` option.

Usage

```
sigma-distance datafile spacefile [options]
```

Options:

`-v|--verbose`: Select informative output.

Both files have to be basisfiles (see *basisfile*, sect. 4.1).

History

`sigma-distance` was implemented by Cord Hockemeyer based on ideas developed in a discussion with Katja Baumunk and Cornelia Dowling (all Technical University of Braunschweig, Germany).

See also

`basisfile` (Sect. 4.1, p. 23), `distance` (Sect. 2.5, p. 11)

2 *KST Specific Tools*

3 General Tools

3.1 `count` — Print a sequence of numbers

Last changed: 11 April 1996

Synopsis

```
count number
```

Description

`count` prints the sequence of numbers from one to `number` to `stdout`. This program is used in a number of shell scripts.

3.2 `permute` — Print a permuted sequence of numbers

Last changed: 11 April 1996

Synopsis

```
permute number
```

Description

`permute` prints a permutation of the sequence of numbers from one to `number` to `stdout`. This program is used in a number of shell scripts.

3 *General Tools*

4 File Formats

4.1 basisfile — Format of basisfiles (v1.0)

Last changed: 11 July 1998

Description

A `basisfile` is an ASCII file describing the basis of a knowledge space. It has the following format:

The first line contains the number of items in the knowledge domain.

The second line contains the number of basis elements.

The following lines contain the basis elements building a matrix where the columns describe the items and the rows describe the basis elements. In each cell of this matrix there is a '0' if the basis element does not contain the item, a '1' if it is a clause for the item and a '2' otherwise.

For any set of knowledge states, a basis according to this specification can be computed with the basis (Sect. 2.1, p. 7) program.

Version information

This manpage describes version v1.0 of the basisfile format. The format changes in v2.0 by additional format and meta information header lines (see `srbifile` (5S)).

See also

`basis` (Sect. 2.1, p. 7), `patternfile` (Sect. 4.2, p. 24) , `spacefile` (Sect. 4.3, p. 25) , `srbifile` (5S)

4.2 **patternfile** — Format of answer pattern files (v1.0)

Last changed: 04 June 1999

Description

Answer pattern files have the same format as knowledge space files. There are two different possibilities to store a `patternfile` — as an ASCII file or as a binary file. Both types describe knowledge spaces in a very similar manner:

ASCII file format

The first line contains the number of items in the knowledge domain. The second line contains the number of answer patterns.

The following lines contain the knowledge states building a matrix where the columns describe the items and the rows describe the answer patterns. In each cell of this matrix there is a '1' if the answer pattern does contain the item, and a '0' otherwise.

Binary file format

The file contains a sequence of `long integer` numbers. The first two numbers give the number of items and the number of knowledge states. The following `long integers` build bitsets, one per answer pattern. A bitset consists of as many `long integers` as are needed to represent the item set. This number of `long integers` needed can be computed as $(\text{ItemNo} + \text{BitsPerLong} - 1) / \text{BitsPerLong}$ where `BitsPerLong` is the machine specific number of bits used to store a `long integer` number.

Warning

Using a binary `patternfile` on different hardware platforms may produce unexpected results since there may be different byte orders and therefore different bit orders.

Version information

This manpage describes version v1.0 of the `basisfile` format. The format changes in v2.0 by additional format and meta information header lines (see `srbfile` (5S)).

See also

basisfile (Sect. 4.1, p. 23), spacefile (Sect. 4.3, p. 25) , srbifile (5S)

4.3 spacefile — Format of spacefiles (v1.0)

Last changed: 04 June 1998

Description

There are two different possibilities to store a *spacefile* — as an ASCII file or as a binary file. Both types describe knowledge spaces in a very similar manner:

ASCII file format

The first line contains the number of items in the knowledge domain. The second line contains the number of knowledge states.

The following lines contain the knowledge states building a matrix where the columns describe the items and the rows describe the knowledge states. In each cell of this matrix there is a '1' if the knowledge state does contain the item, and a '0' otherwise.

Binary file format

The file contains a sequence of `long` integer numbers. The first two numbers give the number of items and the number of knowledge states. The following `long` integers build bitsets, one per knowledge state. A bitset consists of as many `long` integers as are needed to represent the item set. This number of `long` integers needed can be computed as $(\text{ItemNo} + \text{BitsPerLong} - 1) / \text{BitsPerLong}$ where `BitsPerLong` is the machine specific number of bits used to store a `long` integer number.

Warning

Using a binary *spacefile* on different hardware platforms may produce unexpected results since there may be different byte orders and therefore different bit orders.

Version information

This manpage describes version v1.0 of the basisfile format. The format changes in v2.0 by additional format and meta information header lines (see srbifile (5S)).

See also

basisfile (Sect. 4.1, p. 23), patternfile (Sect. 4.2, p. 24) , srbifile (5S)