Project *Surmise Relations between Tests*[*]

SRBT Tools User Manual

Susanne Poetzi and Gudrun Wesiak

August 22, 2002

Institut für Psychologie
Karl–Franzens–Unversität Graz, Austria

[*]The project is financed by the Austrian National Science Fund (FWF) under the project number P12726–SOZ granted to Dietrich Albert and WIlhelm Schappacher.

This text was produced with LaTeX 2$_\varepsilon$. A PostScript version was produced with `dvips` and a PDF version with PDFLaTeX.

Contents

1 File Formats **1**

1.1 Filetypes 1
1.2 basisfile 4
1.3 datafile 5
1.4 disjpartitionfile 6
1.5 ihypothesis 7
1.6 partitionfile 8
1.7 patternfile 8
1.8 spacefile 9
1.9 srbifile 10
1.10 structurefile 11
1.11 testrelation 13

2 SRBI Specific Tools **14**

2.1 combine-equ-items 14
2.2 combine-two-items 15
2.3 complete-srbi 16
2.4 gs-closure 16
2.5 g-constr 17
2.6 parallel-items 18
2.7 srwt 19
2.8 srxt 20

3 Tools for data evaluation **21**

3.1 count-data 21
3.2 count-data-rm 22
3.3 count-patterns 23
3.4 del-equ-data 24
3.5 delete-not-ans 25
3.6 elim 26
3.7 elim-frequ 27
3.8 elim-perc 28
3.9 elim-stud 28
3.10 elim4pat 29
3.11 patt-statistics 30

4 Simulating students answers **32**
4.1 learning-sim 32
4.2 noisy-learn-sim 33

5 Tools for partitions and tests **36**
5.1 connex-part 36
5.2 part-properties 37
5.3 random-part 38
5.4 srbi-part2srbt 39
5.5 test-properties 40

6 General Tools **42**
6.1 new2old 42
6.2 old2new 43
6.3 pat2data 43

1 File Formats

1.1 Filetypes — SRBT File formats

Last changed: 5 May 1999

Synopsis

#SRBT `[version] [structtype] [encoding] [endian] [wordsize] [comment]`

Description

This manpage describes extensions of the old fileformats (see e.g. spacefile (5K), basisfile (5K)). The main extension lies within the definition of additional content types.

Usage

#SRBT `[version] [structtype] [encoding] [endian] [wordsize] [comment]`

`#SRBT` This string at the very beginning of the first line of a file denotes that the file is for use with the `libsrbt` or `libsrbi` libraries.

version Version number of the file format. currently: `v2.0`

structtype This string describes the type of the data contained in the file. Ten types of files are currently supported:
`basis`
`space`
`structure`
`relation` (surmise relation between items)

`data`
`disjpartition` (disjoint partition)
`partition`
`patterns`
`testrelation` (surmise relations between tests)
`i_hypothesis` (partial hypotheses on surmise relations between items)

Since basis, relations, patterns, partitions, are always stored as ASCII text files, the following three, optional file header fields are only valid for data, spaces, and structures.

encoding The `encoding` information specifies whether the data are stored in `ASCII` or in `binary` form.

endian The `endian` information specifies (for binary files only) whether the storing computer has a `LITTLE` or a `BIG` endian processor. Default is `BIG` endian.

wordsize States are stored as a multiple of `wordsize` bits. The default value is 32. Specification of `wordsize` requires a specification of `endian`.

comment The possibility to specify a comment is primarily provided for use with the ASCII files. The first line may contain a comment separated by a hash sign (#). Subsequent lines beginning with a hash sign (#) may also contain comments.

For **binary** files, the format header line and optional comments are closed by a `NUL` (i.e. `0x00`) character.

Additional Information for Items and Students/Tests

It is possible to store non-ambiguous identification numbers for each item and each student or test in the file. This numbers will be called 'item information numbers' and 'line information numbers' in the following. The storage of these numbers is useful especially in cases where the order of items is changed, where items are removed or combined. The information lines must have the following structure (For a detailed description of the structure and interpretation of the below mentioned matrices see the following chapters of the respective filetypes) :

Item information:

`#*` (column in the structure matrix): (non-ambiguous item number(s), seperated by a blank)

Line information:

#+ (line in the structure matrix): (non-ambiguous line number(s), seperated by a blank)

Examples:

```
#* 4:  4
#* 5:  6
#* 6:  7
```

means, that in the forth column of the matrix are the relations for item number 4, in the fith column for the item number 6 and in the 6th column for the item number 7. Item number 5 has been removed.

```
#* 1:  2 3
```

means, that in the first column of the matrix there are the relations for items number 2 and 3 (e.g. if the items 2 and 3 are equivalent).

The same is for line numbers.

ATTENTION: Currently counting of lines and columns starts with 0!!!

If a file is created automatically, additional comment lines are added, which should simplify counting of columns with large numbers of items.

Remarks

The new *SRBT* file format was developed in order to ensure that users do not mingle different file type specifications. This used to become a major problem with users having only little computer experience.

If *SRBT* files are created manually, one should provide as many header information as possible.

Any tools should be able to read files in the old format also. If a file of the wrong type (in the new format) is passed to a program, it may either be rejected (with an appropriate error message) or converted (optionally issuing an additional warning that a file of wrong type was passed over).

See also

basisfile (Sect. 1.2, p. 4), patternfile (Sect. 1.7, p. 8) , srbifile (Sect. 1.9, p. 10) , spacefile (Sect. 1.8, p. 9)

1.2 basisfile — Format of basisfiles (v2.0)

Last changed: 12 May 2000

Description

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

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

`#SRBT` v2.0 basis

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

The third line contains the number of basis elements.

After these three lines you can add comments and identification numbers for items and states in the basis (for a detailed description see Filetypes (Sect. 1.1, p. 1)).

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 (1K) program.

Version information

This manpage describes version v2.0 of the `basisfile` format. In version v1.0 no headerlines and no additional comments and information to items and state numbers are possible. Files of the old (v1.0) and new (v2.0) filetypes can be converted into each other with the programs new2old (see *new2old*, sect. 6.1) and old2new (see *old2new*, sect. 6.2).

See also

basis (1K), spacefile (Sect. 1.8, p. 9) , new2old (Sect. 6.1, p. 42) , old2new (Sect. 6.2, p. 43)

1.3 datafile — Format of datafiles (v2.0)

Last changed: 17 May 2000

Description

There are two different possibilities to store a `datafile` — as an ASCII file or as a binary file.

ASCII file format

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 data ASCII

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

The third line contains the number of answer patterns.

After these three lines you can add comments and identification numbers for items and students in the data structure (for a detailed description see Filetypes (Sect. 1.1, p. 1)).

The following lines contain the answer patterns of students where the columns describe the items and the rows describe the answer patterns of the students: In each cell of this matrix there is a '1' if the student answered the respective item correctly, and a '0', if the answer is wrong.

Binary file format

The first line is the again the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)). The file contains a sequence of `long integer` numbers. The first two numbers give the number of items and the number of answer patterns. 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 `(ItemNo + BitsPerLong - 1) / BitsPerLong` where `BitsPerLong` is the machine specific number of bits used to store a `long integer` number.

If you convert ASCII to binary files, all additional information about items and lines are lost.

Warning

Using a binary `datafile` 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 v2.0 of the `datafile` format. The format changes from v1.0 by additional format and meta information header lines (see spacefile (5K)).

See also

structurefile (Sect. 1.10, p. 11), spacefile (Sect. 1.8, p. 9) , new2old (Sect. 6.1, p. 42) , old2new (Sect. 6.2, p. 43)

1.4 disjpartitionfile — Format of disjoint partitionfiles (v2.0)

Last changed: 17 May 2000

Description

This manpage describes the format and header of a disjoint partition of items into tests.

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 disjpartition

The second line contains the number of items in the domain.

The third line contains the number of tests in the partition.

After these three lines you can add comments and identification numbers for items and tests in the partition (for a detailed description see Filetypes (Sect. 1.1, p. 1)).

The following lines contain the partition of items into different tests where the columns describe the items and the rows describe tests of the partition. Each line of the matrix correspons to a test, in each cell of this matrix there is a '1' if the item belongs to the tests in the respective line, and a '0', if the item does not belong to the test. In disjoint partition files each item must exactly belong to one test. It is not possible, that items belong to more than one tests or that items do not belong to any test at all.

See also

partitionfile (Sect. 1.6, p. 8)

1.5 ihypothesis — File formats for partial input of relations between items

Last changed: 18 May 2000

This manpage describes the format and header of files including surmise relations between items, if not all relations on the whole domain are known.

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 i_hypothesis

The second line contains the number of items.

After these two lines you can add comments and identification numbers for the items in the file.

In this file, a matrix including the surmise relations between items is stored, but not all relations between all items need to be known. The files have the same structure as srbifiles (see *srbifile*, sect. 1.9) including the surmise relations between items, but if you do not know the relation between two special items, you can enter a 'n' on the respective position.

Example: You know, that there is no surmise relation from item i to item j, but you do not know, if there is a surmise relation from item j to item i (if j is a prerequisite from i) (iSj?): you enter a '0' in the j-th column and i-th line of the matrix and a 'n' on the j-th line and i-th column of the matrix.

Warning

Currently this filetype is not supported by any program of this software package.

See also

Filetypes (Sect. 1.1, p. 1), srbifile (Sect. 1.9, p. 10)

1.6 partitionfile — Format of partitionfiles (v2.0)

Last changed: 17 May 2000

Description

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 partition

The second line contains the number of items in the domain.

The third line contains the number of tests in the partition.

After these three lines you can add comments and identification numbers for items and tests in the partition (for a detailed description see Filetypes (Sect. 1.1, p. 1)).

The following lines contain the partition of items into different tests. Each line of the matrix corresponds to a test, in each cell of this matrix there is a '1' if the item belons to the tests in the respective line, and a '0', if the item does not belong to the test.

In partition files it is possible, that items belong to more than one test or that items do not belong to any test at all.

See also

disjpartitionfile (Sect. 1.4, p. 6)

1.7 patternfile — Format of patternfiles (v2.0)

Last changed: 17 May 2000

Description

The patterns files store the answers of students to a set of items, distinguishing between right and wrong answered items and un-answered items.

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 patterns

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

The third line contains the number of answer patterns.

After these three lines you can add comments and identification numbers for items and students in the patterns structure (for a detailed description see Filetypes (Sect. 1.1, p. 1)).

The following lines contain the answer patterns of students: In each cell of this matrix there is a '1' if the student answered the respective item correctly, and a '0', if the answer is wrong, and a 'x', if the student did not answer the item at all.

See also

datafile (Sect. 1.3, p. 5), pat2data (Sect. 6.3, p. 43)

1.8 spacefile — Format of spacefiles (v2.0)

Last changed: 17 May 2000

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 is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 space ASCII

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

The third line contains the number of states in the knowledge space.

After these three lines you can add comments and identification numbers for items and states in the basis (for a detailed description see Filetypes (Sect. 1.1, p. 1)).

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 first line is the again the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)). 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 `(ItemNo + BitsPerLong - 1) / BitsPerLong` where `BitsPerLong` is the machine specific number of bits used to store a `long integer` number.

If you convert ASCII to binary files, all additional information about items and lines are lost.

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 v2.0 of the `spacefile` format. The format changes from v1.0 by additional format and meta information header lines (see spacefile (5K)).

See also

basisfile (Sect. 1.2, p. 4), new2old (Sect. 6.1, p. 42) , old2new (Sect. 6.2, p. 43)

1.9 srbifile — File formats for relationfiles between items

Last changed: 27 July 1999

This manpage describes the format and header of files for surmise relations between items (In this documentation currently either called relationfile or srbifile!).

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)): `#SRBT` v2.0 relation

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

After these two lines you can add comments and identification numbers for items in the srbifile.

The following lines describe the surmise relations between items in form of a matrix: The matrix includes a '1' in line i and column j, iff there is a surmise relation from i to j (jSi, j is surmisable from i, j is a prerequisite of i), a '0' else. Because of the reflexitivity of the relation, in the main diagonal are always '1's. If you want to complete a surmise relation matrix because of transitivity properties, use the program complete-srbi (see *complete-srbi*, sect. 2.3).
Remember: jSi means: item j is surmiseable form item i, j is a prerequisite from i.

Warning

Please note that the introductionary string `#SRBT` is used for `srbifiles`, too.

See also

Filetypes (Sect. 1.1, p. 1), complete-srbi (Sect. 2.3, p. 16)

1.10 structurefile — Format of structurefiles (v2.0)

Last changed: 8 May 2000

Description

This page describes the format and header of files for storage of arbitray structures for working with knowledge spaces (parts of spaces, data,...).

There are two different possibilities to store a `structurefile` — as an ASCII file or as a binary file. Both types describe arbitrary structures for working with knowledge spaces:

ASCII file format

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 structure ASCII

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

The third line contains the number of states/data sets... in structure

After these two lines you can add identification numbers for items and/or states (for a detailed description of the format of the information lines, see Filetypes (Sect. 1.1, p. 1)).

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 `(ItemNo + BitsPerLong - 1) / BitsPerLong` where `BitsPerLong` is the machine specific number of bits used to store a `long integer` number.

If you convert ASCII to binary files, all additional information about items and lines are lost.

Warning

Using a binary `structurefile` 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 v2.0 of the structurefile format. The format changes from v1.0 by additional format and meta information header lines (see spacefile (5K)).

See also

Filetypes (Sect. 1.1, p. 1), spacefile (Sect. 1.8, p. 9).

1.11 testrelation — File format for surmise relations between tests

Last changed: 17 May 2000

This manpage describes the format and header of files for surmise relations between tests. Surmise relation, leftcovering surmise relation and rightcovering surmise relation between tests are included.

The first line is the header-line containg information about version and filetype (see Filetypes (Sect. 1.1, p. 1)):

#SRBT v2.0 testrelation

The second line contains the number of tests

After these two lines you can add comments and identification numbers for the tests in the relationfile.

In this file, three diffent matrices are stored:
The first matrix describes the surmise relations between tests: The matrix includes a '1' in line i and column j, iff there is a surmise relation from test nr. i to test nr. j: jSi, a '0' else. Because of the reflexitivity of the relation, in the main diagonal are always '1's.
The second matrix includes the rightcovering surmise relations between tests following the same schema as described above, the third matrix includes the leftcovering surmise relations. If this file is created automatically, there are included comment lines between the three matrices.

See also

Filetypes (Sect. 1.1, p. 1), srbifile (Sect. 1.9, p. 10) , srbi-part2srbt (Sect. 5.4, p. 39) Brandt, S., Albert, D., Hockemeyer, C. (1999). Surmise relations between tests - preliminary results of the mathematical modelling. Electronic Notes in Discrete Mathematics, 2.

2 SRBI Specific Tools

2.1 combine-equ-items — Combine equivalent items

Last changed: 25 April 2000

Synopsis

combine-equ-items `srbifile outputfile`

Description

`combine-equ-items` combines equivalent items in a relation matrix of surmise relation between items. Two items a, b are called equivalent here, iff aSb and bSa (S denotes the surmise relation between items).

If, for example, the first and the second item in a srbi-matrix are equivalent, the second line and the second column in the matrix is removed. In the 'item information lines' the information, that in the first line/column there are the relations for the (original) first and second item, are added. Before testing, which items are equivalent, the surmise relation matrix is completed because of reflexitivity and transitivity properties (see complete-srbi (Sect. 2.3, p. 16)).
Outputfile is a srbi-file, where all equivalent items have been combined.

Usage

combine-equ-items `srbifile outputfile`

See also

srbifile (Sect. 1.9, p. 10), complete-srbi (Sect. 2.3, p. 16)

2.2 combine-two-items — Combine two items

Last changed: 25 April 2000

Synopsis

combine-two-items `srbi-file outputfile item1 item2`

Description

`combine-two-items` combines two items in a surmise relation matrix. You can choose, which items should be combined, but the given items have to be either parallel or equivalent.

Two items a, b are called equivalent here, iff aSb and bSa.

They are called parallel, iff there exist two items c, d different from a, b in the relation matrix, so that cSa, cSb and aSd, bSd (S denotes the surmise relation between items).
If, for example, the first and the second item in a srbi-matrix are equivalent/parallel and should be combined, the second line and the second column in the matrix is removed. In the item information lines the information, that in the first line/column there are the relations for the (original) first and second item, are added. Before testing, which items are equivalent/parallel, the surmise relation matrix is completed because of reflexitivity and transitivity properties (see complete-srbi (Sect. 2.3, p. 16)).
Outputfile is a srbi-file, where the given items have been combined.

Usage

combine-two-items `srbi-file outputfile item1 item2`
`item1` and `item2` are integer numbers between 0 and number of items - 1. These two items have to be either parallel or equivalent!
ATTENTION: Counting starts with '0'!

See also

srbifile (Sect. 1.9, p. 10), complete-srbi (Sect. 2.3, p. 16) , combine-equ-items (Sect. 2.1, p. 14)

2.3 complete-srbi — Complete a srbi-matrix because of transitivity and reflexitivity properties of the surmise relation between items

Last changed: 25 April 2000

Synopsis

complete-srbi `srbifile outputfile`

Description

`complete-srbi` completes a surmise relation matrix between items because of transitivity and reflexitivity properties of the surmise relation between items. This means, that (1) in the main diagonal of the matrix, there have to be only '1's, because each item is in surmise relation with itself. (2) if there is a '1' in column j and line i (meaning jSi) and in column i and line k (meaning iSk), the program sets a '1' in column j and line k, too (because then jSk is valid). Outputfile is a srbifile (Sect. 1.9, p. 10) including the completed matrix.

Usage

complete-srbi `srbifile outputfile`

See also

srbifile (Sect. 1.9, p. 10)

2.4 gs-closure — Compute the closure under intersection

Last changed: 20 April 2000

Synopsis

gs-closure `[Options] structurefile outputfile`

Description

`gs-closure` computes the closure under intersection of a family of sets.

`gs-closure` uses the algorithm developed by B. Ganter. The program is equivalent to the program s-closure (see s-closure (5K)), but with large spaces and a large number of items it has the advantage, that it needs less computer memory, because not all calculated states have to be kept in memeory, the algorithm needs only one state to calculate the next in a lexicografic order. This is a great advantage especially with very large states and a large amount of items.

Usage

gs-closure `[Options] structurefile outputfile`

Filetype of outputfile will be structurefile.
Options are:

`-a|--ascii:` Use ASCII format for `outputfile` (see *structurefile*, sect. 1.10).

`-b|--binary:` Use binary format for `outputfile` (see *structurefile*, sect. 1.10).

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

See also

datafile (Sect. 1.3, p. 5), structurefile (Sect. 1.10, p. 11)
Bernhard Ganter and Klaus Reuter (1991): Finding all closed sets: A general approach. Order, (8), pp. 283-290

2.5 g-constr — Compute the closure under union

Last changed: 20 April 2000

Synopsis

g-constr `[Options] structurefile spacefile`

Description

`g-constr` computes the closure under union of a family of sets. Input can be a basisfile (see_see(basisfile,5S) or an arbitrary structure. Result is a knowledge space in a spacefile (see_see(spacefile,5S). `g-constr` uses the algorithm developed by B. Ganter. Its main usage is the construction of knowledge spaces from their bases. The program is equivalent to the program constr (see_ext(constr,5K)), but with large spaces and a large number of items it has the advantage, that it needs much less memory.

Usage

g-constr `[Options] structurefile spacefile`
Options are:

- `-a:` Use ASCII format for `spacefile` (see *spacefile*, sect. 1.8).
- `-b:` Use binary format for `spacefile` (see *spacefile*, sect. 1.8).
- `-v:` Select informative output.

See also

datafile (Sect. 1.3, p. 5), spacefile (Sect. 1.8, p. 9) , basisfile (Sect. 1.2, p. 4) , constr (5K)
Bernhard Ganter and Klaus Reuter (1991): Finding all closed sets: A general approach. Order, (8), pp. 283-290

2.6 parallel-items — Look for parallel items in a surmise relation matrix

Last changed: 26 April 2000

Synopsis

parallel-items `srbifile`

Description

`parallel-items` look for parallel items in a surmise relation matrix for parallel items. (Two items a and b are called parallel here, iff there exists two items c, d different from a, b in the relation matrix, so that cSa, cSb and aSd, bSd (S denotes the surmise relation between items).)
The results are printed to stdout.

Usage

parallel-items `srbifile`

Bugs

The definition of paralellity of items does not include special cases!!! Give a new definition!!!

See also

srbifile (Sect. 1.9, p. 10), combine-two-items (Sect. 2.2, p. 15)

2.7 srwt — Look for surmise relations within tests

Last changed: 26 April 2000

Synopsis

srwt `srbifile partitionfile outputfile`

Description

`srwt` takes a matrix with surmise relations between items and a partition of these items into tests. It looks for all surmise relations within tests (both items have to be members of the same test). The results are printed to stdout, a srbi-matrix is written to outputfile, where all surmise relations across tests (between items of different tests) are set '0'.

Usage

srwt `srbifile partitionfile outputfile`
Filetype of outputfile is `srbifile`.

See also

srbifile (Sect. 1.9, p. 10), srxt (Sect. 2.8, p. 20)

2.8 srxt — Look for surmise relations across tests

Last changed: 26 April 2000

Synopsis

srxt `srbifile partitionfile outputfile`

Description

`srxt` takes a matrix with surmise relations between items and a partition of these items into tests. It looks for all surmise relations across tests (both items have to be members of different tests). The results are printed to stdout, a srbi-matrix is written to outputfile, where all surmise relations within tests (between items of the same tests) are set '0'.

Usage

srxt `srbifile partitionfile outputfile`
Filetype of outputfile is `srbifile`.

See also

srbifile (Sect. 1.9, p. 10), srwt (Sect. 2.7, p. 19)

3 Tools for data evaluation

3.1 count-data — Count the frequencies of answer-patterns in a given data-matrix

Last changed: 12 April 2000

Synopsis

count-data `datafile new-datafile [spacefile]`

Description

`count-data` counts the frequencies of answer-patterns in a given data-matrix. The data-matrix must be stored in a datafile (Sect. 1.3, p. 5) in either binary or ASCII format. The frequencies of each line in the data-matrix are printed to stdout, and, optionally, if a spacefile (Sect. 1.8, p. 9) is given, a tag, which shows, if the pattern is element of the space. In the new datafile all repeated patterns are deleted, each pattern is stored only once together with the line number(s) of this pattern in the original datafile.
Example:
Assume the following 4 answer patterns:
100
110
100
111
The program includes line information with the number of the original line, the answer patterns occured (Attention: counting starts with '0'!): The new datafile will include the answer patterns in the following form:

```
#* 0:  0 2
#* 1:  1
#* 2:  3
```

100
110

111
In this file you can see, that the answer pattern in the first line (no. 0!) occures twice in the original file (in the first and third line), the others occured only once.

Usage

count-data `datafile new-datafile [spacefile]`
Spacefile is an optional parameter.

Bugs

Currently only ASCII output of the new datafile is possible.

See also

datafile (Sect. 1.3, p. 5), spacefile (Sect. 1.8, p. 9) , count-patterns (Sect. 3.3, p. 23)

3.2 count-data-rm — Count the frequencies of answer-patterns in a given data-matrix and delete patters with only ’0’s or ’1’s

Last changed: 11 May 2000

Synopsis

count-data-rm `datafile new-datafile [spacefile]`

Description

`count-data-rm` counts the frequencies of answer-patterns in a given data-matrix. The programm works analogous the program count-data (see *count-data*, sect. 3.1), but all answer patterns, which answered all items correct (11111...11) and those, who answered all items wrong (00000...00) are stored only once. The output is analogous to the output of count-data (Sect. 3.1, p. 21).

Example:
Assume the following 5 answer patterns:
100
110
111
111
100
The program includes line information with the number of the original line, the answer patterns occured (Attention: counting starts with '0'!): The new datafile will include the answer patterns in the following form, all the patterns, where no item or all items were answered, are written only once:

```
#* 0:  0 4
#* 1:  1
#* 2:  2
```

100
110
111

Usage

count-data-rm `datafile new-datafile [spacefile]`
Spacefile is an optional parameter.

Bugs

Currently only ASCII output of the new datafile is possible.

See also

datafile (Sect. 1.3, p. 5), spacefile (Sect. 1.8, p. 9) , count-data (Sect. 3.1, p. 21)

3.3 count-patterns — Count the frequencies of answer-patterns in a given patterns-matrix

Last changed: 12 April 2000

Synopsis

count-patterns `patternfile new-patternfile [spacefile]`

Description

`count-patterns` counts the frequencies of answer-patterns in a given matrix of answer patterns. The matrix has to be stored in form of a patternfile (Sect. 1.7, p. 8). The frequencies of all different answer patterns are printed to stdout, and, optionally, if a spacefile (Sect. 1.8, p. 9) is given, a tag, which shows, if it is possible that the pattern is element of the space. In the new patternfile all repeated patterns are deleted, each pattern is stored only once together with the line number(s) of this pattern in the original patternfile (compare count-data (Sect. 3.1, p. 21)).

Usage

count-patterns `patternfile new-patternfile [spacefile]`
Spacefile is an optional parameter.

See also

patternfile (Sect. 1.7, p. 8), spacefile (Sect. 1.8, p. 9) , count-data (Sect. 3.1, p. 21)

3.4 del-equ-data — Delete equal answer patterns

Last changed: 21 April 2000

Synopsis

del-equ-data `datafile outputfile`

Description

`del-equ-data` deletes equal answer patterns out of a data-file. Item numbers of the original file are kept, but line-infos are removed. Outputfile is a datafile (Sect. 1.3, p. 5) again, where each answer-pattern occurs only once. If you want to keep line information, use the program count-data (see *count-data*, sect. 3.1).

Usage

del-equ-data `datafile outputfile`
Filetype of outputfile is datafile.

Bugs

Currently only ASCII output is possible.

See also

datafile (Sect. 1.3, p. 5), count-data (Sect. 3.1, p. 21)

3.5 delete-not-ans — Delete patterns that did not answer certain items

Last changed: 13 April 2000

Synopsis

delete-not-ans `patternfile datafile no_items`

Description

`delete-not-ans` calculates for each item, how many students gave an answer to the item ('0' or '1' in the matrix of answer patterns, not 'x'). The given number of the items, which have been answered by fewest students, are deleted. Afterwards all patterns, which did not give an answer to all the remaining items are deleted. Result is a datafile, with `no_items` less items than the original file.

Usage

delete-not-ans `patternfile datafile no_items`

See also

patternfile (Sect. 1.7, p. 8), patt-statistics (Sect. 3.11, p. 30)

3.6 elim — Eliminate answer patterns that do not fit into closedness under union and intersection

Last changed: 26 April 2000

Synopsis

elim `[Options] datafile outputfile`

Description

`elim` eliminates answer patterns, that do not fit into closedness under union and intersection, which is an important property of quasi ordinal knowledge spaces. `elim` looks, which answer patterns in the datafile do most frequently contradict the assumption, that the union(intersection) of two arbitrary answer patterns is again an answer pattern. The algorithm eliminates the patterns for which the quotient 'number of contradictions' divided through 'frequency of the pattern' is a maximum. It works recursive, until no contradictions to the closedness under union and intersection are left. Before using the original elim-program, the frequencies of the different patterns have to be counted. This is automatically done by the program count-data (Sect. 3.1, p. 21).

The program prints to stdout, which answer patterns are removed. The remaining patterns are written to the outputfile, which is a `datafile` again (see datafile (Sect. 1.3, p. 5)).

Usage

elim `[Options] datafile outputfile`
Options are:

- `-a:` Use ASCII format for `spacefile` (see *spacefile*, sect. 1.8).
- `-b:` Use binary format for `spacefile` (see *spacefile*, sect. 1.8).
- `-v:` Select informative output.

See also

datafile (Sect. 1.3, p. 5), count-data (Sect. 3.1, p. 21) Brandt, S., Poetzi, S., Albert, D., (in preperation). Simulating noisy knowledge spaces by means of the base. Manuscript in preperation.

3.7 elim-frequ — Eliminate answer patterns that do not fit into closedness under union and intersection

Last changed: 26 April 2000

Synopsis

elim-frequ `datafile outputfile frequency-percentage`

Description

`elim-frequ` eliminates answer patterns in a datafile, that do not fit into closedness under union and intersection, which is an important property of quasi ordinal knowledge spaces. The algorithm of this program is the same as in the program elim (Sect. 3.6, p. 26), but all the answer patterns, which occur more often than the given percentage of all answer patterns are not eliminated, even when they include most contradictions (e.g. 100 patterns, percentage = 5, all patterns, that occur more often than 5 times are not deleted). Before using the original `elim-frequ` program, the frequencies of the different patterns have to be counted. This is automatically done by the program count-data (Sect. 3.1, p. 21). The program prints to stdout, which answer patterns are removed. The remaining patterns are written to the outputfile, which is a datafile (Sect. 1.3, p. 5) again.

Usage

elim-frequ `datafile outputfile frequency-percentage`
Filetype of outputfile is `datafile`.

See also

datafile (Sect. 1.3, p. 5), count-data (Sect. 3.1, p. 21) , elim (Sect. 3.6, p. 26)

3.8 elim-perc — Eliminate answer patterns that do not fit into closedness under union and intersection

Last changed: 26 April 2000

Synopsis

elim-perc `datafile outputfile contradiction-percentage`

Description

`elim-perc` eliminates answer patterns in a datafile, that do not fit into closedness under union and intersection, which is an important property of quasi ordinal knowledge spaces. The algorithm of this program is the same as in the program elim (Sect. 3.6, p. 26), but the algorithm is not stopped, when no contradictions are left, but when less than a given percentage of contradictions is reached. (E.g. you give a percentage of 5 with 100 patterns, the algorithm stops, if for all patterns less than 5 contradictions are left). Before using the original `elim-perc` program, the frequencies of the different patterns have to be counted. This is automatically done by the program count-data (Sect. 3.1, p. 21). The program prints to stdout, which answer patterns are removed. The remaining patterns are written to the outputfile, which is a datafile (Sect. 1.3, p. 5) again.

Usage

elim-perc `datafile outputfile contradiction-percentage`
Filetype of outputfile is `datafile`.

See also

datafile (Sect. 1.3, p. 5), count-data (Sect. 3.1, p. 21) , elim (Sect. 3.6, p. 26)

3.9 elim-stud — Eliminate answer patterns that do not fit into closedness under union and intersection

Last changed: 26 April 2000

Synopsis

elim-stud `datafile outputfile no_patterns-left`

Description

`elim-stud` eliminates answer patterns in a datafile, that do not fit into closedness under union and intersection, which is an important property of quasi ordinal knowledge spaces. The algorithm of this program is the same as in the program elim (Sect. 3.6, p. 26), but the algorithm stops, when approximatively the given number of different patterns is left. Before using the original `elim-stud` program, the frequencies of the different patterns have to be counted. This is automatically done by the program count-data (Sect. 3.1, p. 21). The program prints to stdout, which answer patterns are removed. The remaining patterns are written to the outputfile, which is a datafile (Sect. 1.3, p. 5) again.

Usage

elim-stud `datafile outputfile no_patterns-left`
Filetype of outputfile is `datafile`.

See also

datafile (Sect. 1.3, p. 5), count-data (Sect. 3.1, p. 21) , elim (Sect. 3.6, p. 26)

3.10 elim4pat — Eliminate patterns that do not fit into closedness under union and intersection

Last changed: 26 April 2000

Synopsis

elim4pat `patternfile outputfile`

Description

`elim4pat` eliminates answer patterns in a patternfile, that do not fit into closedness under union and intersection, which is an important property of quasi ordinal knowledge spaces. `elim4pat` works with the same algorithm as the program elim (see *elim*, sect. 3.6), but it works with answer patterns, which include the possibility of correct answered ('1'), wrong answered('0') or not answered questions('x') (see patternfile (Sect. 1.7, p. 8)).

For this program, we defined the union(u) and intersection (s) of patterns in the following way:
(111 000 xxx) u (10x 10x 10x) = (111 10x 1xx)
(111 000 xxx) s (10x 10x 10x) = (10x 000 x0x)

Before using the original elim4pat-program, the frequencies of the different patterns have to be counted. This is automatically done by the program count-patterns (see *count-patterns*, sect. 3.3). The program prints to stdout, which answer patterns are removed. The remaining patterns are written to the outputfile, which is a patternfile (see *patternfile*, sect. 1.7) again.

Usage

elim4pat `patternfile outputfile`

See also

patternfile (Sect. 1.7, p. 8), count-patterns (Sect. 3.3, p. 23) , elim (Sect. 3.6, p. 26)

3.11 patt-statistics — Calculate how many students gave an answer to a selected group of items

Last changed: 12 April 2000

Synopsis

patt-statistics `patternfile`

Description

`patt-statistics` counts, how many students gave an answer to all items ('0' or '1', not 'x'). Next the item which has been answered by fewest students is deleted and the programm counts, how many students answered all remaining items. Again the item, which has been answered by fewest students is deleted and so on. The results are printed to stdout.
Finally the program calculates how many items have to be deleted to get (1) the maximal number of students, that answered all items, (2) the maximal number of items, that has been answered by all students and (3) the maximum of the product 'items*students, which answered all items'.

The program can be used the find the appropriate item number for the program delete-not-ans (Sect. 3.5, p. 25).

Usage

patt-statistics `patternfile`

See also

patternfile (Sect. 1.7, p. 8), delete-not-ans (Sect. 3.5, p. 25)

4 Simulating students answers

4.1 learning-sim — Simulating students using the learning path model

Last changed: 12 April 2000

Synopsis

learning-sim `basisfile outputfile probabilitiesfile no_students`

Description

`learning-sim` simulates the knowledge states of students using the learning path model. From a given basis the possible learning paths are calculated. The probabilities file includes (1) the probabilities for the quantity of items learned (e.g. the probabilitiy for learning half of the items is higher than the probability for learning all items....) and (2) the probabilites for learning item x rather than item y.

The probabilities file has the following form:

prob. for learning no item
prob. for learning 1 item
prob. for learning 2 items
.....
prob. for learning all items
if you learn an item: prob. for learning item 1
prob. for learning item 2
.....
prob. for learning item q

Call the number of items in the basis q, then the first q+1 probabilities have to sum to ’1’ and the second q probabilities have to sum to ’1’! The randomly (with the help of the

given probabilities) chosen learning paths are written to stdout. The knowledge states for all simulated students are written to the outputfile, which is a datafile (Sect. 1.3, p. 5). This resulting sets of states is equal to the space or a subset of the space, which would result from the basis, but the states which include a certain quantity of items, that is more probably to be learned, occur more often, and the items that are more probably learned, occur more often in the learning paths. No careless errors or lucky guesses are included to the file, if you want to include careless errors and/or lucky guesses, use program noisy-learn-sim (Sect. 4.2, p. 33) .

Usage

learning-sim `basisfile outputfile probabilitiesfile no_students`
Filetype of outputfile is a `datafile` in ASCII format.

Bugs

Currently only ASCII output of the new datafile is possible.

See also

datafile (Sect. 1.3, p. 5), basisfile (Sect. 1.2, p. 4) , noisy-learn-sim (Sect. 4.2, p. 33)
Brandt, S., Poetzi, S., Albert, D. (in preperation). Simulation knowledge spaces by means of the base. Manuscript in preperation.

4.2 noisy-learn-sim — Simulating students making careless errors and lucky guesses using the learning path model

Last changed: 20 April 2000

Synopsis

noisy-learn-sim `basisfile outputfile probabilitiesfile no_students beta eta`

Description

`noisy-learn-sim` simulates the knowledge states of students using the learning path model. From a given basis the possible learning paths are calculated. The probabilities file includes (1) the probabilities for the quantity of items learned (e.g. the probabilitiy for learning half of the items is higher than the probability for learning all items....) and (2) the probabilites for learning item x rather than item y.

The probabilities file has the following form:

prob. for learning no item
prob. for learning 1 item
prob. for learning 2 items
.....
prob. for learning all items
if you learn an item: prob. for learning item 1
prob. for learning item 2
.....
prob. for learning item q

Call the number of items in the basis q, then the first q+1 probabilities have to sum to '1' and the second q probabilities have to sum to '1'! The randomly (with the help of the given probabilities) chosen learning paths are written to stdout. After choosing the states in which the students are according to the given probabilities in the probability file, careless errors and lucky guesses are simulated. The value beta is the probability for a lucky guess, the value eta for a careless error (both numbers have to be values between 0 and 1). The simulated answer patterns are written to a datafile (Sect. 1.3, p. 5).

Usage

noisy-learn-sim `basisfile outputfile probabilitiesfile no_students beta eta`
Filetype of outputfile is a `datafile` in ASCII format.

Bugs

Currently only ASCII output of the new datafile is possible.

See also

datafile (Sect. 1.3, p. 5), basisfile (Sect. 1.2, p. 4) , learning-sim (Sect. 4.1, p. 32) Brandt, S., Poetzi, S., Albert, D. (in preperation). Simulation noisy knowledge spaces by means of the base. Manuscript in preperation.

5 Tools for partitions and tests

5.1 connex-part — Produce a connex disjoint partition of items into tests

Last changed: 21 April 2000

Synopsis

connex-part `srbifile outputfile`

Description

`connex-part` creates a connex disjoint partition of items into tests. The program tries to generate as few tests as possible (only one item per test would be a trivial solution that is always possible).
A test is called 'connex' here, if for each item 'n' in the test there exists another item 'm' in the same test, so that nSm or mSn ('S' denotes the surmise relation between items).
A partition is called connex, if all tests in the partition are connex.
Outputfile is a disjoint partition file (see disjpartitionfile (Sect. 1.4, p. 6)).

Usage

connex-part `srbifile outputfile`
Filetype of outputfile is `disjpartitionfile`.

See also

disjpartitionfile (Sect. 1.4, p. 6)

5.2 part-properties — Investigate the properties of a partition of items into tests

Last changed: 20 April 2000

Synopsis

part-properties `srbifile partitionfile`

Description

`part-properties` investigates properties of a partition of items into tests using the surmise relations between the items.
The following properties are investigated:

- Is the whole partition connex?
- Is the whole partition transitive?
- Is the whole partition antisymmetric?

The results are printed to stdout.

Usage

part-properties `srbifile partitionfile`

Bugs

The properties right-left- and totalcoveringness of the whole partition have to be defined theoretically and included into the program.

See also

srbifile (Sect. 1.9, p. 10), partitionfile (Sect. 1.6, p. 8) , test-properties (Sect. 5.5, p. 40)
Brandt, S., Albert, D., Hockemeyer, C. (in press). Surmise relations between tests - mathematical considerations. Discrete Applied Mathematics.

5.3 random-part — Produce a random disjoint partition of items into tests

Last changed: 21 April 2000

Synopsis

random-part `partition-kind no_items no_tests outputfile`

Description

`random-part` produces a random disjoint partition of items into tests. There are five possibilities for choosing the properties of the resulting disjoint partition:

- The number of tests can be chosen randomly
- A maximal number of tests can be specified
- The number of tests can be specified exactly
- Each test must have the same number of items (only possible, if the number of items is a multiple of the number of tests)
- A minimal number of items per test can be requested

Outputfile is a disjoint partition file (see *disjpartitionfile*, sect. 1.4).

Usage

random-part `partition-kind no_items no_tests outputfile`

Possibilities for the kind of partition are: Create a random disjoint partition with

`-r:` a randomly chosen number of tests.

`-m:` a given maximal number of tests.

`-t:` a given number of tests.

`-e:` an equal number of items per test.

`-i<n>:` a minimal number of 'n' items per test.

Filetype of outputfile is disjoint partition file (see *disjpartitionfile*, sect. 1.4).

Bugs

Choosing the kind of partition '-r', a value for a number of tests has to be entered, which is not used in the program.

See also

disjpartitionfile (Sect. 1.4, p. 6)

5.4 srbi-part2srbt — Calculate the surmise relations between tests out of surmise relations between items and a partition into tests

Last changed: 11 May 2000

Synopsis

srbi-part2srbt `srbifile partitionfile testrelation`

Description

`srbi-part2srbt` calculates the surmise relations between tests out of surmise relations between items and a partition of items into tests. The surmise relations between tests, right-and leftcovering surmise relations are calculated. The results are printed to stdout, and the resulting surmise relations are written to a testrelationfile (see testrelation (Sect. 1.11, p. 13)). The relations are written in form of three matrices, one for surmise relations, one for right- and one for leftcovering surmise relations: A '1' in line i and column j means: Test(i) and Test(j) are in surmise relation from Test(i) to Test(j). (Test(j) S Test(i))(S denotes the surmise relation between tests). The same goes for right- and leftcovering surmise relations.
Remember: Two Tests (i) and (j) are in surmise relation from Test(j) to Test(i), iff there is an item a in Test(j) and an item b in Test(i), so that b can surmised from a.

Usage

srbi-part2srbt `srbifile partitionfile testrelation`

See also

testrelation (Sect. 1.11, p. 13), srbifile (Sect. 1.9, p. 10) , partitionfile (Sect. 1.6, p. 8) , test-properties (Sect. 5.5, p. 40)
Brandt, S., Albert, D., Hockemeyer, C. (1999). Surmise relations betweeen tests - preliminary results of the mathematical modelling. Electronic Notes in Discrete Mathematics, 2.

5.5 test-properties — Investigate the tests of a given partition for the properties left- right- and total-coveringness as well as antisymmetry and connectivity

Last changed: 20 April 2000

Synopsis

test-properties `srbifile partitionfile`

Description

`test-properties` investigates properties of tests in a given partition using the surmise relations between the items in the tests. For all possible combinations of tests is investigated:

- Is there a surmise relation between tests?
- Is there a left-right- or total-covering surmise relation between tests?
- Are two tests antisymmetric?
- Is a given test connex?

The results are printed to stdout.

Usage

test-properties `srbifile partitionfile`

See also

srbifile (Sect. 1.9, p. 10), partitionfile (Sect. 1.6, p. 8) , part-properties (Sect. 5.2, p. 37)
Brandt, S., Albert, D., Hockemeyer, C. (1999). Surmise relations betweeen tests - preliminary results of the mathematical modelling. Electronic Notes in Discrete Mathematics, 2.

6 General Tools

6.1 new2old — Change file in new file format (v2.0) to file in old format

Last changed: 8 May 2000

Synopsis

new2old filename-(new-format) filename-(old-format)

Description

`new2old` changes a file in new file format (containing a header line and information numbers) (see new file types (see *Filetypes*, sect. 1.1)) to a file in old format (no header line, no information numbers, this filetype is still needed for some of the 'old' programs, e.g. basis (5K), s-closure (5K)). The type of file, information numbers of items and lines are lost with this transformation.

Usage

new2old filename-new-format filename-old-format

See also

old2new (Sect. 6.2, p. 43), Filetypes (Sect. 1.1, p. 1)

6.2 old2new — Change file in old file format to file in new format (v2.0)

Last changed: 8 May 2000

Synopsis

old2new filename-old-format filename-new-format filetype

Description

`old2new` changes a file in old file format to a file in new format (containing a header line with information of the filetype) (see Filetypes (Sect. 1.1, p. 1)).

Usage

old2new filename-old-format filename-new-format filetype
Filetype is one of the types of files frequently used with knowledge space theory (space, basis, data,..., see Filetypes (Sect. 1.1, p. 1)).

See also

new2old (Sect. 6.1, p. 42), Filetypes (Sect. 1.1, p. 1)

6.3 pat2data — Change a patternfile to a datafile

Last changed: 26 April 2000

Synopsis

pat2data `patternfile datafile`

Description

`pat2data` changes a patternfile to a datafile. If for an item no answer is given (x in the patternfile), in the datafile, 0 is included (the answer is wrong).

Usage

pat2data `patternfile datafile`

See also

patternfile (Sect. 1.7, p. 8), datafile (Sect. 1.3, p. 5)