KST Tools User Manual

Cord Hockemeyer
Cornelia E. Dowling

Institut für Psychologie
Technische Universität Braunschweig
Germany

30 June 1996

Contents

I. Overview **1**

1. Some Introductory Remarks **1**

1.1. Availability 1

1.2. Conventions Used in this Report 2

2. Program List **2**

2.1. Different Representations 2

2.2. Working with Rules 2

2.3. Informations 2

2.4. Combining Different Knowledge Spaces 3

2.5. Comparing Different Knowledge Spaces 3

2.6. Changing Knowledge Spaces 3

2.7. Creating special Knowledge Spaces 3

2.8. Working with Surmise Relations 4

2.9. Validation Using Student Data 4

II. KST Specific Tools **5**

3. basis **5**

4. basis-union **6**

5. constr **6**

6. dep-pairs **7**

7. distance **8**

8. heights **10**

9. mk-empty-space **11**

10. select **12**

11. sigma-distance **12**

III. General Tools **14**

12. count **14**

13. permute **14**

IV. File Formats **15**

14. basisfile **15**

15. patternfile **15**

16. spacefile **16**

Part I.

Overview

1. Some Introductory Remarks

This report documents a number of programs mainly written by Cord Hockemeyer during his time in the research group of Cornelia Dowling at the Department of Psychology, University of Technology at Braunschweig, Germany. Many ideas have also been contributed by other members of this group, namely by Katja Baumunk and Rainer Kaluscha.

This documentation represents a snapshot of the set of programs and their documentation as Cord Hockemeyer and Rainer Kaluscha are moving from Braunschweig to Graz, Austria. Cord Hockemeyer will continue maintaining tools and documentation there (see also Section 1.1 on future availability of programs and documentation). Section 2 gives a short overview of all programs available. For a selected number of programs, more complete documentation can be found in Part II of this document. Part III describes some minor tools applied by other programs, and part IV documents the file formats used.

The work reported herein was financially supported through two project grants given to Cornelia Dowling by the Ministry of Education in Luxemburg an by the Alexander von Humboldt Foundation, and through a post–graduate fellowship given to Cord Hockemeyer by the Government of the German state of Lower Saxony.

1.1. Availability

Since the maintainer of the described collection of tools, Cord Hockemeyer, moves to Graz, the files will be downloadable from there. Please have a look at the URL `http://wundt.kfunigraz.ac.at/hockemeyer/tools.html`, or contact him by e–mail at `Cord.Hockemeyer@kfunigraz.ac.at`.

Some programs of this collection make use of the `libkst` library[1]. This library can be reached through the same address.

[1] A previous version in the form of three separate libraries `libset`, `libspace`, and `libgeneral` was documented by Cornelia E. Dowling and Cord Hockemeyer (*Wissensdiagnose in der beruflichen Ausbildung [Knowledge assessment in professional education].* Technical report, Institut für Psychologie, Technische Universität Braunschweig, Germany, 1995).

1.2. Conventions Used in this Report

In Section 2, there is a note for each program stating the program language and — if appropriate, i. e. in case of scripts — the names of other programs used. There are three languages used: *C*, *C++*, and *bash* (the latter meaning shell scripts using the `bash` or the `ksh`). The scripts may be not portable. For the shell scripts the *C/C++* programs used within which have been written especially for knowledge spaces are 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*.

For those programs documented in more detail in Parts II and III, the documentation is kept in UNIX™ man page style since it is actually a pronted version of the respective man page

2. Program List

2.1. Different Representations

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

2.2. 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)*

2.3. 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)*
- **equivalence:** Compute and print classes of equivalent items in a knowledge space given by its basis. *(C)*
- **heights:** Compute for each item its height within a knowledge space defined as the minimal size of its clauses. *(C)*

2.4. Combining Different Knowledge Spaces

- **basis-union:** 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)*

2.5. Comparing Different Knowledge Spaces

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

2.6. Changing Knowledge Spaces

- **s-closure:** Compute the closure under intersection of a knowledge space (given by its basis). The resulting spaces is also given by its basis. *(bash, using the constr and the* basis *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)*
- **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:** Select randomly a number of knowledge states from a given knowledge space and store it in the format of a knowledge space. *(C)*

2.7. 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:** Just the complement program to *diagonal-basis*. Write a space-file which contains only the empty set and the complete item set as its knowledge states. *(C)*

2.8. Working with Surmise Relations

- **dep-matrix:** Show a surmise relation as a matrix. *(C, link to dep-pairs)*
- **dep-pairs:** 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)*

2.9. Validation Using Student Data

- **rule-errors:** Compute for each pair in a surmise relation the number of students' answer patterns which contradict the pair. *(bash, using the* dep-pairs *program mentioned above)*
- **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:** 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:** 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 Held. In the meantime, it has been extended to the computation of several measures on the invalidity of items and prerequisite relationships. *(C)*
- **clause-inv:** This program computes a measure for the invalidity of item-clause pairs in a given basis with respect to a specified data set of answer patterns. *(C)*

Part II.

KST Specific Tools

3. basis — Computing the basis of a knowledge structure

Last changed: 25 April 1996

Synopsis

basis `[options] structurefile basisfile`

Description

`basis` computes the basis of a knowledge structure. The structure may be stored in a spacefile (Sect. 16) in either binary or ASCII format. The resulting basis is stored in basisfile (Sect. 14) format. The elements of the basis are ordered by their size so it can be used directly with the constr (Sect. 5) program.

Usage

basis `[Options] structurefile basisfile`

Options are:

`-a|-ascii`: Use ASCII format for structurefile (see *spacefile*, sect. 16).

`-b|-binary`: Use binary format for structurefile (see *spacefile*, sect. 16).

Only one of these options may be used.

See also

basisfile (Sect. 14) , spacefile (Sect. 16) , constr (Sect. 5)

4. basis-union — Computing the basis representing the union of two knowledge spaces

Last changed: 19 April 1996

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. 14).

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. 16) , basisfile (Sect. 14) , basis (Sect. 3)

5. constr — Computing the closure under union

Last changed: 16 May 1996

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. 16) from their bases (see *basisfile*, sect. 14).

Usage

constr `[Options] structure space`

`structure` may be in spacefile (Sect. 16) or basisfile (Sect. 14) 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.

Options are:

`-a|-ascii`: Use ASCII format for `spacefile` (see *spacefile*, sect. 16).

`-b|-binary`: Use binary format for `spacefile` (see *spacefile*, sect. 16).

`-c|-complement`: Compute the complements of the states before storing the space

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

See also

spacefile (Sect. 16) , basisfile (Sect. 14)

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

6. dep-pairs dep-all dep-matrix — Print dependence information in a surmise relation

Last changed: 27 May 1996

Synopsis

dep-pairs `basisfile`
dep-all `basisfile`
dep-matrix `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 conlude 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 then 40 elements, then the columns of the matrix are seperated by a space, otherwise they are not seperated at all.

See also

basisfile (Sect. 14)

7. distance di invalidity — Computing the distance between two knowledge spaces

Last changed: 04 June 1996

Synopsis

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

Usage

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

Options:

`-a|-ascii`: Assume following files to be in ASCII format.

`-b|-binary`: Assume following files to be in binary 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`)

`-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`). **Not yet implemented!**

`-q|-quiet`: Do not produce informative output.

`-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.

Defaults:

`distance: -binary -dist-only -quiet` (general default)

`di: -ascii -dist-only -table -list`

`invalidity: -binary -item -prereq`

Both files have to be knowledge space files (see *spacefile*, sect. 16) or answer pattern files (see *patternfile*, sect. 15) in the selected format.

History

The 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.

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

See also

spacefile (Sect. 16) , sigma-distance (Sect. 11)

8. 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. 14)

9. mk-empty-space — Create an almost empty knowledge space

Last changed: 11 April 1996

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. 16).

`-b|-binary`: Use binary format for `spacefile` (see *spacefile*, sect. 16).

Only one of these options may be used.

See also

spacefile (Sect. 16)

10. select — Select randomly a set of knowledge states from a knowledge space

Last changed: 11 April 1996

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. 16) 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. 16).

See also

spacefile (Sect. 16) , srand (3) , time (2)

11. 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. 14).

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. 14) , distance (Sect. 7)

Part III.

General Tools

12. 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.

13. 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.

Part IV.

File Formats

14. basisfile — Format of basisfiles

Last changed: 04 June 1996

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 rows describe the items and the columns 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.

See also

patternfile (Sect. 15) , spacefile (Sect. 16)

15. patternfile — Format of answer pattern files

Last changed: 04 June 1996

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

```
(ItemNo + BitsPerLong - 1) / 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.

See also

basisfile (Sect. 14) , spacefile (Sect. 16)

16. spacefile — Format of spacefiles

Last changed: 04 June 1996

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

```
(ItemNo + BitsPerLong - 1) / 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.

See also

basisfile (Sect. 14) , patternfile (Sect. 15)