Package 'MDP2'

June 12, 2026

Type Package

Title Markov Decision Processes (MDPs)

Version 2.2.2.0

Author Lars Relund Nielsen [aut, cre] (ORCID: <https://orcid.org/0000-0002-4802-3071>)

Maintainer Lars Relund Nielsen <lars@relund.dk>

Description Create and optimize (semi) MDPs with discrete time steps and state space. Both hierarchical and ordinary-traditional MDPs can be modeled.

License GPL (>= 3)

URL https://relund.github.io/mdp/, https://github.com/relund/mdp/, http://relund.github.io/mdp/

BugReports https://github.com/relund/mdp/issues

Depends R (>= 4.1.0)

Imports diagram, dplyr, stringr, tidyr, magrittr, methods, purrr, rlang, tibble, Rcpp (>= 0.11.5)

Suggests knitr, Matrix, rmarkdown, testthat (>= 3.0.0), readr, xml2, covr, roxygen2

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-06-12 19:30:14 UTC

Contents

`binaryActionWriter` *Function for writing actions of a HMDP model to binary files. The function defines sub-functions which can be used to define actions saved in a set of binary files. It is assumed that the states have been defined using* `binaryMDPWriter` *and that the id of the states is known (can be retrieved using e.g.* `stateIdxDf`*).*

Description

Binary files are efficient for storing large models. Compared to the HMP (XML) format the binary files use less storage space and loading the model is faster.

Usage

```
binaryActionWriter(
  prefix = "",
  binNames = c("actionIdx.bin", "actionIdxLbl.bin", "actionWeight.bin",
   "actionWeightLbl.bin", "transProb.bin", "transWeight.bin", "transWeightLbl.bin"),
  append = TRUE
)
```

Arguments

prefix	A character string with the prefix added to binNames.
binNames	A character vector of length 5 giving the names of the binary files storing the model.
append	Logical indicating whether should keep the currents actions (default - TRUE) defined or delete them and start over (FALSE).

Details

The returned writer exposes these functions:

- setWeights(labels, ...): sets the labels of the weights used in the actions. labels is a vector of label names. ... is currently ignored. Call this before building the model.
- addAction(label = NULL, sIdx, weights, prob, ...): adds an action. sIdx is the id of the state defining the action. weights must be a vector of action weights. prob is a matrix (sIdx, pr) where the first column contains the id of the transition state; see the description of actionIdx.bin below, where scope is assumed to be 3. ... is currently ignored.
- endAction(): ends an action.
- closeWriter(): closes the writer. Call this when the model description is finished.

Five binary files are created:

- actionIdx.bin: integers defining all actions in the format sIdx scope idx scope idx scope idx -1 sIdx scope idx sIdx corresponds to the index or line number in stateIdx.bin, starting from 0. The following (scope, idx) pairs indicate possible transitions. Scope can take four values:
  - 2: a transition to a child process, at stage zero in the child process.
  - 1: a transition to the next stage in the current process.
  - 0: a transition to the next stage in the father process.
  - 3: a transition to a state specified by its state sIdx.

  For example, if scope = 1 and idx = 2, the transition is to state number 3 at the next stage in the current process. If scope = 3 and idx = 5, the transition is to the state specified at line 6 in stateIdxLbl.bin. This is useful when considering shared child processes.
- actionIdxLbl.bin: character data in the format aIdx label aIdx label .... Here aIdx corresponds to the index or line number in actionIdx.bin, starting from 0. No delimiter is used.
- actionWeight.bin: doubles containing action weights in the format "c1 c2 c3 c1 c2 c3 ...", assuming three weights for each action.
- actionWeightLbl.bin: character data containing the weight labels in the format label1 label2 label3, assuming three weights for each action.
- transProb.bin: doubles containing the transition probabilities defined in actionIdx.bin. The format is "p1 p2 p3 -1 p1 -1 p1 p2 -1 ...". Here -1 indicates that a new action is considered.

Value

A list of functions.

Note

Note all indexes are starting from zero (C/C++ style).

Examples

```
## Use temp dir
wd <- setwd(tempdir())

# Create a small HMDP with two levels
w<-binaryMDPWriter()
w$setWeights(c("Duration","Net reward","Items"))
w$process()
   w$stage()
      w$state(label="M0")
         w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
            w$process()
               w$stage()
                  w$state(label="D")
                     w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
               w$endStage()
               w$stage()
                  w$state(label="C0")
                     w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                     w$endAction()
                     w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
                  w$state(label="C1")
                     w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                     w$endAction()
                     w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
               w$endStage()
               w$stage()
                  w$state(label="C0")
                     w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                     w$endAction()
                  w$endState()
                  w$state(label="C1")
                     w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                     w$endAction()
                  w$endState()
               w$endStage()
            w$endProcess()
         w$endAction()
         w$action(label="A1",weights=c(0,0,0),prob=c(2,0,1))
            w$process()
               w$stage()
                  w$state(label="D")
```

```
                w$action(label="A0",weights=c(0,0,1),prob=c(1,0,1))
                w$endAction()
             w$endState()
          w$endStage()
          w$stage()
             w$state(label="C0")
                w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                w$endAction()
                w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
                w$endAction()
             w$endState()
          w$endStage()
          w$stage()
             w$state(label="C0")
                w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                w$endAction()
             w$endState()
             w$state(label="C1")
                w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                w$endAction()
                w$action(label="A1",weights=c(0,10,5),prob=c(0,0,0.5,0,1,0.5))
                w$endAction()
             w$endState()
          w$endStage()
       w$endProcess()
    w$endAction()
 w$endState()
 w$state(label="M1")
    w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
       w$process()
          w$stage()
             w$state(label="D")
                w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
                w$endAction()
             w$endState()
          w$endStage()
          w$stage()
             w$state(label="C0")
                w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                w$endAction()
             w$endState()
             w$state(label="C1")
                w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                w$endAction()
             w$endState()
          w$endStage()
          w$stage()
             w$state(label="C0")
                w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                w$endAction()
             w$endState()
             w$state(label="C1")
                w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
```

```
                    w$endAction()
                  w$endState()
                w$endStage()
              w$endProcess()
            w$endAction()
          w$endState()
        w$endStage()
      w$endProcess()
      w$closeWriter()

      ## Info about the binary files (don't have to load the model first)
      getBinInfoStates()
      getBinInfoActions()

      ## reset working dir
      setwd(wd)
```

`binaryMDPWriter` *Function for writing an HMDP model to binary files. The function defines sub-functions which can be used to define an HMDP model saved in a set of binary files.*

Description

Binary files are efficient for storing large models. Compared to the HMP (XML) format the binary files use less storage space and loads the model faster.

Usage

```
binaryMDPWriter(
  prefix = "",
 binNames = c("stateIdx.bin", "stateIdxLbl.bin", "actionIdx.bin", "actionIdxLbl.bin",
  "actionWeight.bin", "actionWeightLbl.bin", "transProb.bin", "externalProcesses.bin",
   "transWeight.bin", "transWeightLbl.bin"),
  getLog = TRUE
)
```

Arguments

`prefix`	A character string with the prefix added to `binNames`.
`binNames`	A character vector giving the names of the binary files storing the model.
`getLog`	Output log text.

Details

The returned writer exposes these functions:

- `setWeights(labels, ...)`: sets the labels of the weights used in the actions. `labels` is a vector of label names. `...` is currently ignored. Call this before building the model.

- process(): starts a (sub)process. It may also be used to specify a traditional MDP using matrices in MDPtoolbox style. In that style, P is a list of matrices, one per action, each of size $S x S$ where S is the number of states. Each used row must sum to one, or all entries in a row must be zero if unused. R is a matrix of size $S x A$, where A is the number of actions, and D is a matrix of size $S x A$ with durations. If D is omitted, all durations are assumed to be 1.
- endProcess(): ends a (sub)process.
- stage(label = NULL): starts a stage. label is currently unused in the binary format.
- endStage(): ends a stage.
- state(label = NULL): starts a state and returns, invisibly, the state id. That id can later be referenced with scope 3.
- endState(): ends a state.
- action(scope = NULL, id = NULL, pr = NULL, prob = NULL, weights, transWeights = NULL, label = NULL, end = FALSE, ...): starts an action. weights must be a vector of action weights. transWeights must contain transition weights ordered by transition, with all transition weight labels for the first transition followed by all labels for the second transition, and so on. Transition probabilities can be entered in two ways:
  1. prob contains triples (scope, id, pr).
  2. id and pr are vectors of equal length. If scope is omitted, all scopes default to 1.

  See the description of actionIdx.bin below. If end = TRUE, calling endAction() is not necessary. ... is currently ignored.
- endAction(): ends an action. Do not use this if end = TRUE was used when the action was specified.
- includeProcess(prefix, label = NULL, weights, prob, termStates, transWeights = NULL): includes an external process. External processes are loaded into memory only when needed, which helps with large models. prefix is the external process prefix. weights must be a vector of action weights, and prob must contain triples (scope, idx, pr); see the description of actionIdx.bin below. termStates must specify the number of states at the last stage in the external process. Inside an includeProcess ... endIncludeProcess block, you must specify the father jump actions of the last stage in the external process. The external process is represented by its first and last stage together with its jump actions. The function returns, invisibly, the state ids of the first stage in the external process, which can later be referenced with scope 3.
- endIncludeProcess(): ends an includeProcess block.
- closeWriter(): closes the writer. Call this when the model description is finished.

Ten binary files are created:

- stateIdx.bin: integers defining all states in the format "n0 s0 -1 n0 s0 a0 n1 s1 -1 n0 s0 a0 n1 s1 a1 n2 s2 -1 n0 s0 ...". Here -1 indicates that a new state is considered.
- stateIdxLbl.bin: character data in the format sIdx label sIdx label .... Here sIdx corresponds to the index or line number in stateIdxLbl.bin, starting from 0. No delimiter is used.
- actionIdx.bin: integers defining all actions in the format sIdx scope idx scope idx scope idx -1 sIdx scope idx sIdx corresponds to the index or line number in stateIdx.bin, starting from 0. The following (scope, idx) pairs indicate possible transitions. Scope can take four values:

  - 2: a transition to a child process, at stage zero in the child process.
  - 1: a transition to the next stage in the current process.
  - 0: a transition to the next stage in the father process.
  - 3: a transition to a state specified by its state sIdx.

  For example, if scope = 1 and idx = 2, the transition is to state number 3 at the next stage in the current process. If scope = 3 and idx = 5, the transition is to the state specified at line 6 in stateIdxLbl.bin. This is useful when considering shared child processes.

- actionIdxLbl.bin: character data in the format aIdx label aIdx label .... Here aIdx corresponds to the index or line number in actionIdx.bin, starting from 0. No delimiter is used.
- actionWeight.bin: doubles containing action weights in the format "c1 c2 c3 c1 c2 c3 ...", assuming three weights for each action.
- actionWeightLbl.bin: character data containing the weight labels in the format label1 label2 label3, assuming three weights for each action.
- transProb.bin: doubles containing transition probabilities defined in actionIdx.bin. The format is "p1 p2 p3 -1 p1 -1 p1 p2 -1 ...". Here -1 indicates that a new action is considered.
- externalProcesses.bin: character data containing links to external processes in the format stageStr prefix stageStr prefix .... Here stageStr corresponds to the stage index, for example n0 s0 a0 n1, of the stage corresponding to the first stage in the external process, and prefix is the external process prefix. No delimiter is used.
- transWeight.bin: doubles containing transition weights in the format "t11 t12 t21 t22 -1 ...", assuming two transition weights for each transition and two transitions in the first action.
- transWeightLbl.bin: character data containing the transition weight labels.

Value

A list of functions.

Note

Note all indexes are starting from zero (C/C++ style).

Examples

```
## Use temp dir
wd <- setwd(tempdir())

# Create a small HMDP with two levels
w<-binaryMDPWriter()
w$setWeights(c("Duration","Net reward","Items"))
w$process()
   w$stage()
      w$state(label="M0")
         w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
            w$process()
               w$stage()
                  w$state(label="D")
```

```
            w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
            w$endAction()
         w$endState()
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
            w$endAction()
            w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
            w$endAction()
         w$endState()
         w$state(label="C1")
            w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
            w$endAction()
            w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
            w$endAction()
         w$endState()
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
            w$endAction()
         w$endState()
         w$state(label="C1")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
            w$endAction()
         w$endState()
      w$endStage()
   w$endProcess()
w$endAction()
w$action(label="A1",weights=c(0,0,0),prob=c(2,0,1))
   w$process()
      w$stage()
         w$state(label="D")
            w$action(label="A0",weights=c(0,0,1),prob=c(1,0,1))
            w$endAction()
         w$endState()
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
            w$endAction()
            w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
            w$endAction()
         w$endState()
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
            w$endAction()
         w$endState()
         w$state(label="C1")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
```

```
                    w$endAction()
                    w$action(label="A1",weights=c(0,10,5),prob=c(0,0,0.5,0,1,0.5))
                    w$endAction()
                 w$endState()
              w$endStage()
           w$endProcess()
        w$endAction()
     w$endState()
     w$state(label="M1")
        w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
           w$process()
              w$stage()
                 w$state(label="D")
                    w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
                    w$endAction()
                 w$endState()
              w$endStage()
              w$stage()
                 w$state(label="C0")
                    w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                    w$endAction()
                 w$endState()
                 w$state(label="C1")
                    w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                    w$endAction()
                 w$endState()
              w$endStage()
              w$stage()
                 w$state(label="C0")
                    w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                    w$endAction()
                 w$endState()
                 w$state(label="C1")
                    w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                    w$endAction()
                 w$endState()
              w$endStage()
           w$endProcess()
        w$endAction()
     w$endState()
  w$endStage()
w$endProcess()
w$closeWriter()

## Info about the binary files (don't have to load the model first)
getBinInfoStates()
getBinInfoActions()

## reset working dir
setwd(wd)
```

convertBinary2HMP *Convert a HMDP model stored in binary format to a* hmp *(XML) file. The function simply parse the binary files and create* hmp *files using the* hmpMDPWriter().

Description

Convert a HMDP model stored in binary format to a hmp (XML) file. The function simply parse the binary files and create hmp files using the hmpMDPWriter().

Usage

```
convertBinary2HMP(
  prefix = "",
 binNames = c("stateIdx.bin", "stateIdxLbl.bin", "actionIdx.bin", "actionIdxLbl.bin",
    "actionWeight.bin", "actionWeightLbl.bin", "transProb.bin"),
  out = paste0(prefix, "converted.hmp"),
  duration = 1,
  getLog = TRUE
)
```

Arguments

prefix	A character string with the prefix which will be added to the binary files.
binNames	A character vector of length 7 giving the names of the binary files storing the model.
out	The name of the HMP file (e.g. r.hmp).
duration	Weight number storing the duration (NULL if none).
getLog	Output log text.

Value

NULL (invisible).

Note

Note all indexes are starting from zero (C/C++ style).

See Also

convertHMP2Binary().

Examples

```
## Set working dir
fDir <- system.file("models", package = "MDP2")
wd <- setwd(tempdir())
## Convert the machine example to a hmp file
prefix1 <- paste0(fDir,"/machine1_")
getBinInfoStates(prefix1)
convertBinary2HMP(prefix1, duration = NULL, out = "machine1_converted.hmp")
# have a look at the hmp file
cat(readr::read_file("machine1_converted.hmp"))

## Convert the machine example hmp file to binary files
convertHMP2Binary(file = paste0(fDir,"/machine1.hmp"), prefix = "machine_cov_")
getBinInfoStates(prefix = "machine_cov_")
## Convert the machine example with a single dummy node to a hmp file
#convertBinary2HMP("machine2_")  # error since using scope = 3 not supported in hmp files

## Reset working dir
setwd(wd)
```

---

convertHMP2Binary *Convert a HMDP model stored in a hmp (xml) file to binary file format.*

Description

The function simply parse the hmp file and create binary files using the `binaryMDPWriter()`.

Usage

```
convertHMP2Binary(file, prefix = "", getLog = TRUE)
```

Arguments

`file`	The name of the HMP file (e.g. `r.hmp`).
`prefix`	A character string with the prefix which will be added to the binary files.
`getLog`	Output log text.

Value

NULL (invisible).

Note

Note all indexes are starting from zero (C/C++ style).

See Also

`binaryMDPWriter()`.

Examples

```
## Set working dir
fDir <- system.file("models", package = "MDP2")
wd <- setwd(tempdir())
## Convert the machine example to a hmp file
prefix1 <- paste0(fDir,"/machine1_")
getBinInfoStates(prefix1)
convertBinary2HMP(prefix1, duration = NULL, out = "machine1_converted.hmp")
# have a look at the hmp file
cat(readr::read_file("machine1_converted.hmp"))

## Convert the machine example hmp file to binary files
convertHMP2Binary(file = paste0(fDir,"/machine1.hmp"), prefix = "machine_cov_")
getBinInfoStates(prefix = "machine_cov_")
## Convert the machine example with a single dummy node to a hmp file
#convertBinary2HMP("machine2_")  # error since using scope = 3 not supported in hmp files

## Reset working dir
setwd(wd)
```

---

getBinInfoActions *Info about the actions in the HMDP model under consideration.*

Description

Info about the actions in the HMDP model under consideration.

Usage

```
getBinInfoActions(
  prefix = "",
  labels = TRUE,
  fileA = "actionIdx.bin",
  filePr = "transProb.bin",
  fileW = "actionWeight.bin",
  fileLabelW = "actionWeightLbl.bin",
  fileLabelA = "actionIdxLbl.bin"
)
```

Arguments

prefix	A character string with the prefix added to til binary files.
labels	Should labels be extracted.
fileA	The binary file containing the description of actions.
filePr	The binary file containing the description of transition probabilities.
fileW	The binary file containing the description of weights.
fileLabelW	The binary file containing the weight labels.
fileLabelA	The binary file containing the action labels.

Value

A data frame with the information. Scope string contain the scope of the transitions and can be 4 values:

- 0: A transition to the next stage in the father process,
- 1: A transition to next stage in the current process,
- 2: A transition to a child process (stage zero in the child process),
- 3: A transition to the state with `sId = idx` is considered.

The index string denote the index (id is scope = 3) of the state at the next stage.

Note

The model don't have to be loaded, i.e only read the binary files. The state id (`sId`) will not be the same as in the loaded model!

Examples

```
## Use temp dir
wd <- setwd(tempdir())

# Create a small HMDP with two levels
w<-binaryMDPWriter()
w$setWeights(c("Duration","Net reward","Items"))
w$process()
   w$stage()
      w$state(label="M0")
         w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
            w$process()
               w$stage()
                  w$state(label="D")
                     w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
               w$endStage()
               w$stage()
                  w$state(label="C0")
                     w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                     w$endAction()
                     w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
                  w$state(label="C1")
                     w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                     w$endAction()
                     w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
               w$endStage()
               w$stage()
                  w$state(label="C0")
```

```
                  w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                  w$endAction()
               w$endState()
               w$state(label="C1")
                  w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                  w$endAction()
               w$endState()
            w$endStage()
         w$endProcess()
      w$endAction()
      w$action(label="A1",weights=c(0,0,0),prob=c(2,0,1))
         w$process()
            w$stage()
               w$state(label="D")
                  w$action(label="A0",weights=c(0,0,1),prob=c(1,0,1))
                  w$endAction()
               w$endState()
            w$endStage()
            w$stage()
               w$state(label="C0")
                  w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                  w$endAction()
                  w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
                  w$endAction()
               w$endState()
            w$endStage()
            w$stage()
               w$state(label="C0")
                  w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                  w$endAction()
               w$endState()
               w$state(label="C1")
                  w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                  w$endAction()
                  w$action(label="A1",weights=c(0,10,5),prob=c(0,0,0.5,0,1,0.5))
                  w$endAction()
               w$endState()
            w$endStage()
         w$endProcess()
      w$endAction()
   w$endState()
   w$state(label="M1")
      w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
         w$process()
            w$stage()
               w$state(label="D")
                  w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
                  w$endAction()
               w$endState()
            w$endStage()
            w$stage()
               w$state(label="C0")
                  w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
```

```
                    w$endAction()
                 w$endState()
                 w$state(label="C1")
                    w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                    w$endAction()
                 w$endState()
              w$endStage()
              w$stage()
                 w$state(label="C0")
                    w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                    w$endAction()
                 w$endState()
                 w$state(label="C1")
                    w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                    w$endAction()
                 w$endState()
              w$endStage()
           w$endProcess()
        w$endAction()
     w$endState()
  w$endStage()
w$endProcess()
w$closeWriter()

## Info about the binary files (don't have to load the model first)
getBinInfoStates()
getBinInfoActions()

## reset working dir
setwd(wd)
```

`getBinInfoStates` *Info about the states in the binary files of the HMDP model under consideration.*

Description

Info about the states in the binary files of the HMDP model under consideration.

Usage

```
getBinInfoStates(
  prefix = "",
  labels = TRUE,
  stateStr = TRUE,
  fileS = "stateIdx.bin",
  labelS = "stateIdxLbl.bin"
)
```

Arguments

prefix	A character string with the prefix added to til binary files.
labels	Should labels be extracted.
stateStr	Should state strings be extracted. If false then add columns (n0, s0, a0, ...) where n0 the index of the stage at level 0, s0 the index of the state and a0 the index of the action. If the HMDP has more than one level columns index (d1, s1, a1, ...) are added.
fileS	The binary file containing the description of states.
labelS	The binary file containing the state labels.

Value

A data frame with the information.

Note

The model don't have to be loaded, i.e only read the binary files. The state id (sId) will not be the same as in the loaded model!

---

getHypergraph *Return the (parts of) state-expanded hypergraph*

Description

The function is useful together with plotHypergraph().

Usage

```
getHypergraph(mdp, ...)
```

Arguments

mdp	The MDP loaded using loadMDP().
...	Arguments passed to getInfo().

Value

A list representing the hypergraph with two elements: a tibble nodes and a tibble hyperarcs. hyperarcs stores actionWeights, trans, and pr as list-columns of vectors. transWeights is a list-column of matrices with one row per transition and one column per transition-weight namespace.

See Also

plotHypergraph() and plot.HMDP().

Examples

```
## Set working dir
wd <- setwd(system.file("models", package = "MDP2"))

#### A finite-horizon replacement problem ####
mdp<-loadMDP("machine1_")
plot(mdp)
plot(mdp, actionColor = "label")  # colors based on labels
plot(mdp, transLabels = "state")  # label transitions with target state labels
plot(mdp, transLabels = "prob")  # label transitions with transition probabilities
plot(mdp, actionColor = "label", stateLabel = "sId|label")  # state labels are 'sId | label'
plot(mdp, stateLabel = "sIdx|label", radx = 0.01)  # adjust radx in states
plot(mdp, stateLabel = "label", actionWLabel = "none", actionLabel = "label",
     transLabels = "sId", radx = 0.01)

scrapValues <- c(30, 10, 5, 0)  # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, "Net reward" , termValues = scrapValues)
plot(mdp, actionColor = "policy")  # highlight optimal policy
plot(mdp, actionsVisible = "policy", stateLabel = "weight")  # show only optimal policy


#### An infinite-horizon maintenance problem ####
mdp<-loadMDP("hct611-1_")
plot(mdp)  # plot the first two stages
plot(mdp, actionColor = "label")  # colors based on labels
plot(mdp, actionColor = "label", stateLabel = "sId|label")  # state labels are 'sId | label'
runPolicyIteAve(mdp,"Net reward","Duration")
plot(mdp, actionColor = "policy")  # highlight optimal policy
plot(mdp, actionsVisible = "policy")  # show only optimal policy


#### An infinite-horizon hierarchical replacement problem ####
library(magrittr)
mdp<-loadMDP("cow_")
hgf <- getHypergraph(mdp)
# modify labels
dat <- hgf$nodes %>%
   dplyr::mutate(label = dplyr::case_when(
      label == "Low yield" ~ "L",
      label == "Avg yield" ~ "A",
      label == "High yield" ~ "H",
      label == "Dummy" ~ "D",
      label == "Bad genetic level" ~ "Bad",
      label == "Avg genetic level" ~ "Avg",
      label == "Good genetic level" ~ "Good",
      TRUE ~ "Error"
   ))
# assign nodes to grid ids
dat$gId[1:3]<-85:87
dat$gId[43:45]<-1:3
getGId<-function(process,stage,state) {
   if (process==0) start=18
```

```
   if (process==1) start=22
   if (process==2) start=26
   return(start + 14 * stage + state)
}
idx<-43
for (process in 0:2)
   for (stage in 0:4)
      for (state in 0:2) {
         if (stage==0 & state>0) break
         idx<-idx-1
         #cat(idx,process,stage,state,getGId(process,stage,state),"\n")
         dat$gId[idx]<-getGId(process,stage,state)
      }
hgf$nodes <- dat
# modify labels
dat <- hgf$hyperarcs %>%
   dplyr::mutate(label = dplyr::case_when(
      label == "Replace" ~ "R",
      label == "Keep" ~ "K",
      label == "Dummy" ~ "D",
      TRUE ~ "Error"
   ),
   col = dplyr::case_when(
      label == "R" ~ "deepskyblue3",
      label == "K" ~ "darkorange1",
      label == "D" ~ "black",
      TRUE ~ "Error"
   ),
   lwd = 0.5,
   label = ""
   )
hgf$hyperarcs <- dat
# plot hypergraph
oldpar <- par(mai = c(0, 0, 0, 0))
plotHypergraph(gridDim = c(14, 7), hgf, cex = 0.8, radx = 0.02, rady = 0.03)
par(oldpar)


## A simple finite-horizon MDP with action and transition weights
prefix <- file.path(tempdir(), "plot_transition_rewards_")
w <- binaryMDPWriter(prefix)
w$setWeights("Cost")
w$setTransWeights(c("Reward", "Disease"))
w$process()
   w$stage()
      w$state(label = "S1")
         w$action(
            label = "A1", weights = 2, id = c(1), pr = c(1),
            transWeights = c(20, 0.3), end = TRUE
         )
         w$action(
            label = "A2", weights = 1, id = c(0, 1), pr = c(0.3, 0.7),
            transWeights = c(25, 0.4, 15, 0.2), end = TRUE
```

```
        )
      w$endState()
   w$endStage()
   w$stage()
      w$state(label = "S2")
         w$action(
            label = "A3", weights = 3, id = c(0, 1, 2), pr = c(0.5, 0.3, 0.2),
            transWeights = c(0, 0.05, 12, 0.2, 30, 0.8), end = TRUE
         )
         w$action(
            label = "A4", weights = 2, id = c(1, 2), pr = c(0.6, 0.4),
            transWeights = c(22, 0.35, 27, 0.7), end = TRUE
         )
      w$endState()
      w$state(label = "S3")
         w$action(
            label = "A5", weights = 1, id = c(0, 1), pr = c(0.4, 0.6),
            transWeights = c(5, 0, 16, 0.25), end = TRUE
         )
         w$action(
            label = "A6", weights = 4, id = c(0, 1, 2), pr = c(0.1, 0.3, 0.6),
            transWeights = c(14, 0.15, 21, 0.45, 29, 1), end = TRUE
         )
      w$endState()
   w$endStage()
   w$stage()
      w$state(label = "S4", end = TRUE)
      w$state(label = "S5", end = TRUE)
      w$state(label = "S6", end = TRUE)
   w$endStage()
w$endProcess()
w$closeWriter()

mdp <- loadMDP(prefix, getLog = FALSE)
plot(mdp, actionColor = "label", transLabels = "weights", actionWLabel = "weight",
     radx = 0.005, rady = 0.01)

## Reset working dir
setwd(wd)
```

`getInfo` *Information about the MDP*

Description

Information about the MDP

Usage

```
getInfo(
```

```
  mdp,
  sId = 1:ifelse(mdp$timeHorizon < Inf, mdp$states, mdp$states + mdp$founderStatesLast) -
    1,
  stateStr = NULL,
  stageStr = NULL,
  withList = TRUE,
  withDF = TRUE,
  dfLevel = "state",
  asStringsState = TRUE,
  asStringsActions = FALSE,
  withHarc = FALSE
)
```

Arguments

mdp	The MDP loaded using loadMDP().
sId	The id of the state(s) considered.
stateStr	A character vector containing the index of the state(s) (e.g. "n0,s0,a0,n1,s1"). Parameter sId are ignored if not NULL.
stageStr	A character vector containing the index of the stage(s) (e.g. "n0,s0,a0,n1"). Parameter sId and idxS are ignored if not NULL.
withList	Output info as a list lst.
withDF	Output the info as a data frame.
dfLevel	If withDF and equal "state" the data frame contains a row for each state. If equal "action" the data frame contains a row for each action.
asStringsState	Write state vector as a string; otherwise, output it as columns.
asStringsActions	Write action vectors (weights, transitions and probabilities) as strings; otherwise, output it as columns.
withHarc	Output a hyperarcs data frame. Each row contains a hyperarc with the first column denoting the head (sId), the tails (sId) and the label.

Value

A list containing the list, data frame(s).

Examples

```
## Set working dir
wd <- setwd(tempdir())

# Create the small machine repleacement problem used as an example in L.R. Nielsen and A.R.
# Kristensen. Finding the K best policies in a finite-horizon Markov decision process. European
# Journal of Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011.

## Create the MDP using a dummy replacement node
prefix<-"machine1_"
```

```
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
      w$state(label="replaced")       # v=(3,3)
         w$action(label="Dummy", weights=0, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good", end=TRUE)        # v=(4,0)
      w$state(label="average", end=TRUE)     # v=(4,1)
      w$state(label="not working", end=TRUE) # v=(4,2)
```

```
      w$state(label="replaced", end=TRUE)    # v=(4,3)
   w$endStage()
w$endProcess()
w$closeWriter()

## Load the model into memory
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
scrapValues<-c(30,10,5,0)   # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, w, termValues=scrapValues)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)



# The example given in L.R. Nielsen and A.R. Kristensen. Finding the K best
# policies in a finite-horizon Markov decision process. European Journal of
# Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011,
# does actually not have any dummy replacement node as in the MDP above. The same
# model can be created using a single dummy node at the end of the process.

## Create the MDP using a single dummy node
prefix<-"machine2_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
```

```
      w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
      w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
   w$endState()
w$endStage()
w$stage()   # stage n=2
   w$state(label="good")           # v=(2,0)
      w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
      w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
   w$endState()
   w$state(label="average")        # v=(2,1)
      w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
      w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
   w$endState()
   w$state(label="not working")    # v=(2,2)
      w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
   w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE) # transition to sId=12 (Dummy)
   w$endState()
w$endStage()
w$stage()   # stage n=3
   w$state(label="good")           # v=(3,0)
      w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
      w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
   w$endState()
   w$state(label="average")        # v=(3,1)
      w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
      w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
   w$endState()
   w$state(label="not working")    # v=(3,2)
      w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
      w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE)
   w$endState()
w$endStage()
w$stage()   # stage n=4
   w$state(label="good")        # v=(4,0)
      w$action(label="rep", weights=30, prob=c(1,0,1), end=TRUE)
   w$endState()
   w$state(label="average")     # v=(4,1)
      w$action(label="rep", weights=10, prob=c(1,0,1), end=TRUE)
   w$endState()
   w$state(label="not working") # v=(4,2)
      w$action(label="rep", weights=5, prob=c(1,0,1), end=TRUE)
   w$endState()
w$endStage()
w$stage()   # stage n=5
   w$state(label="Dummy", end=TRUE)        # v=(5,0)
w$endStage()
w$endProcess()
w$closeWriter()

## Have a look at the state-expanded hypergraph
mdp<-loadMDP(prefix)
mdp
plot(mdp)
```

```
getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
runValueIte(mdp, w, termValues = 0)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)


## Reset working dir
setwd(wd)
```

`getPolicy` *Get parts of the optimal policy.*

Description

Get parts of the optimal policy.

Usage

```
getPolicy(
  mdp,
  sId = ifelse(mdp$timeHorizon >= Inf, mdp$founderStatesLast + 1,
   1):ifelse(mdp$timeHorizon >= Inf, mdp$states + mdp$founderStatesLast, mdp$states) - 1,
  stageStr = NULL,
  stateLabels = TRUE,
  actionLabels = TRUE,
  actionIdx = TRUE,
  rewards = TRUE,
  stateStr = TRUE,
  external = NULL,
  ...
)
```

Arguments

mdp	The MDP loaded using loadMDP().
sId	Vector of id's of the states we want to retrieve.
stageStr	Stage string. If specified then find sId based on the stage string.
stateLabels	Add state labels.
actionLabels	Add action labels of policy.
actionIdx	Add action index.
rewards	Add weights calculated for each state.
stateStr	Add the state string for each state.
external	A vector of stage strings corresponding to external processes we want the optimal policy of.
...	Parameters passed on when find the optimal policy of the external processes. Note if external is specified then it must contain stage strings from mdp$external. Moreover you must specify further arguments passed on to runValueIte() used for recreating the optimal policy e.g. the g value and the label for weight and duration. See the vignette about external processes.

Value

The policy (data frame).

Examples

```
## Set working dir
wd <- setwd(tempdir())

# Create the small machine repleacement problem used as an example in L.R. Nielsen and A.R.
# Kristensen. Finding the K best policies in a finite-horizon Markov decision process. European
# Journal of Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011.

## Create the MDP using a dummy replacement node
prefix<-"machine1_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
```

```
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
      w$state(label="replaced")       # v=(3,3)
         w$action(label="Dummy", weights=0, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good", end=TRUE)        # v=(4,0)
      w$state(label="average", end=TRUE)     # v=(4,1)
      w$state(label="not working", end=TRUE) # v=(4,2)
      w$state(label="replaced", end=TRUE)    # v=(4,3)
   w$endStage()
w$endProcess()
w$closeWriter()

## Load the model into memory
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
```

```
scrapValues<-c(30,10,5,0)   # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, w, termValues=scrapValues)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)



# The example given in L.R. Nielsen and A.R. Kristensen. Finding the K best
# policies in a finite-horizon Markov decision process. European Journal of
# Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011,
# does actually not have any dummy replacement node as in the MDP above. The same
# model can be created using a single dummy node at the end of the process.

## Create the MDP using a single dummy node
prefix<-"machine2_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
      w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE) # transition to sId=12 (Dummy)
```

```
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good")        # v=(4,0)
         w$action(label="rep", weights=30, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="average")     # v=(4,1)
         w$action(label="rep", weights=10, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="not working") # v=(4,2)
         w$action(label="rep", weights=5, prob=c(1,0,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=5
      w$state(label="Dummy", end=TRUE)        # v=(5,0)
   w$endStage()
w$endProcess()
w$closeWriter()

## Have a look at the state-expanded hypergraph
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
runValueIte(mdp, w, termValues = 0)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
```

```
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)


## Reset working dir
setwd(wd)
```

`getRPO` *Calculate the retention pay-off (RPO) or opportunity cost for some states.*

Description

The RPO is defined as the difference between the weight of the state when using action `iA` and the maximum weight of the node when using another predecessor different from `iA`.

Usage

```
getRPO(
  mdp,
  w,
  iA,
  sId = ifelse(mdp$timeHorizon >= Inf, mdp$founderStatesLast + 1,
   1):ifelse(mdp$timeHorizon >= Inf, mdp$states + mdp$founderStatesLast, mdp$states) - 1,
  criterion = "expected",
  dur = "",
  rate = 0,
  rateBase = 1,
  discountFactor = NULL,
  g = 0,
  objective = c("max", "min"),
  discountMethod = "continuous",
  stateStr = TRUE
)
```

Arguments

`mdp`	The MDP loaded using `loadMDP()`.
`w`	The label of the weight we calculate RPO for.
`iA`	The action index we calculate the RPO with respect to (same size as `sId`).
`sId`	Vector of id's of the states we want to retrieve.
`criterion`	The Bellman operator shortcut. If `expected` use expected weights, if `discount` use discounted expected weights, if `average` use average expected weights.
`dur`	The label of the duration/time such that discount rates can be calculated.
`rate`	The interest rate.
`rateBase`	The time-horizon the rate is valid over.

discountFactor	The discount rate for one time unit. If specified rate and rateBase are not used to calculate the discount rate.
g	The optimal gain (g) calculated (used if criterion = "average").
objective	Optimize by maximizing ("max") or minimizing ("min") the Bellman value.
discountMethod	Either 'continuous' or 'discrete', corresponding to discount factor exp(-rate/rateBase) or 1/(1 + rate/rateBase), respectively. Only used if discountFactor is NULL.
stateStr	Output the state string.

Value

The RPO (matrix/data frame).

`getSteadyStatePr` *Calculate the steady state transition probabilities for the founder process (level 0).*

Description

Assume that we consider an ergodic/irreducible time-homogeneous Markov chain specified using a policy in the MDP.

Usage

```
getSteadyStatePr(mdp, getLog = FALSE)
```

Arguments

mdp	The MDP loaded using loadMDP().
getLog	Output log text.

Value

A vector with steady state probabilities for all the states at the founder level.

`getWIdx` *Return the index of a weight in the model. Note that index always start from zero (C++ style), i.e. the first weight, the first state at a stage etc has index 0.*

Description

Return the index of a weight in the model. Note that index always start from zero (C++ style), i.e. the first weight, the first state at a stage etc has index 0.

Usage

```
getWIdx(mdp, wLbl)
```

Arguments

`mdp`	The MDP loaded using `loadMDP()`.
`wLbl`	The label/string of the weight.

Value

The index (integer).

`hmpMDPWriter` *Function for writing an HMDP model to a hmp file (XML). The function define sub-functions which can be used to define an HMDP model stored in a hmp file.*

Description

HMP files are in XML format and human readable using e.g. a text editor. HMP files are not suitable for storing large HMDP models since text files are very verbose. Moreover, approximation of the weights and probabilities may occur since the parser writing the hmp file may no output all digits. If you consider large models then use the binary file format instead.

Usage

```
hmpMDPWriter(
  file = "r.hmp",
  rate = 0.1,
  rateBase = 1,
  precision = 1e-05,
  desc = "HMP file created using hmpMDPWriter in R",
  getLog = TRUE
)
```

Arguments

file	The name of the file storing the model (e.g. r.hmp).
rate	The interest rate (used if consider discounting).
rateBase	The time where the rate is taken over, e.g. if the rate is 0.1 and rateBase is 365 days then we have an interest rate of 10 percent over the year.
precision	The precision used when checking if probabilities sum to one.
desc	Description of the model.
getLog	Output log text.

Details

The returned writer exposes these functions:

- setWeights(labels, duration): sets the labels of the weights used in the actions. labels is a vector of label names. duration identifies which label corresponds to duration or time. For example, if the first entry in labels is time, then duration = 1. Call this before building the model.
- setTransWeights(labels): sets the labels of transition-level weights.
- process(): starts a (sub)process.
- endProcess(): ends a (sub)process.
- stage(label = NULL): starts a stage.
- endStage(): ends a stage.
- state(label = NULL): starts a state and returns the state index sIdx.
- endState(): ends a state.
- action(label = NULL, weights, prob, statesNext = NULL, transWeights = NULL): starts an action. weights must be a vector of action weights, and prob must contain triples (scope, idx, pr). scope can take three values:
  - 0: a transition to the next stage in the father process.
  - 1: a transition to the next stage in the current process.
  - 2: a transition to a child process, at stage zero in the child process.

  The idx value denotes the index of the state at the stage considered. For example, if scope = 1 and idx = 2, the transition is to state number 3 at the next stage in the current process, counting from zero. scope = 3 is not supported in the hmp file format. statesNext is the number of states in the next stage of the process and is only needed when there is a transition to the father.
- endAction(): ends an action.
- closeWriter(): closes the writer. Call this when the model description is finished.

Value

A list of functions.

Note

Note all indexes are starting from zero (C/C++ style).

Examples

```
## Use temp dir
wd <- setwd(tempdir())

## Create a small HMDP with two levels
w<-hmpMDPWriter()
w$setWeights(c("Duration","Net reward","Items"), duration=1)
w$process()
  w$stage()
   w$state(label="M0")
     w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
      w$process()
        w$stage()
         w$state(label="D")
           w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
           w$endAction()
         w$endState()
        w$endStage()
        w$stage()
         w$state(label="C0")
           w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
           w$endAction()
           w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
           w$endAction()
         w$endState()
         w$state(label="C1")
           w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
           w$endAction()
           w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
           w$endAction()
         w$endState()
        w$endStage()
        w$stage()
         w$state(label="C0")
           w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1), statesNext=0)
           w$endAction()
         w$endState()
         w$state(label="C1")
           w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1), statesNext=0)
           w$endAction()
         w$endState()
        w$endStage()
      w$endProcess()
     w$endAction()
     w$action(label="A1",weights=c(0,0,0),prob=c(2,0,1))
      w$process()
        w$stage()
         w$state(label="D")
```

```
          w$action(label="A0",weights=c(0,0,1),prob=c(1,0,1))
          w$endAction()
        w$endState()
       w$endStage()
       w$stage()
        w$state(label="C0")
          w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
          w$endAction()
          w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
          w$endAction()
        w$endState()
       w$endStage()
       w$stage()
        w$state(label="C0")
          w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1), statesNext=0)
          w$endAction()
        w$endState()
        w$state(label="C1")
          w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1), statesNext=0)
          w$endAction()
          w$action(label="A1",weights=c(0,10,5),prob=c(0,0,0.5,0,1,0.5), statesNext=0)
          w$endAction()
        w$endState()
       w$endStage()
     w$endProcess()
    w$endAction()
  w$endState()
  w$state(label="M1")
    w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
     w$process()
       w$stage()
        w$state(label="D")
          w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
          w$endAction()
        w$endState()
       w$endStage()
       w$stage()
        w$state(label="C0")
          w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
          w$endAction()
        w$endState()
        w$state(label="C1")
          w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
          w$endAction()
        w$endState()
       w$endStage()
       w$stage()
        w$state(label="C0")
          w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1), statesNext=0)
          w$endAction()
        w$endState()
        w$state(label="C1")
          w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1), statesNext=0)
```

```
          w$endAction()
        w$endState()
       w$endStage()
     w$endProcess()
    w$endAction()
  w$endState()
 w$endStage()
w$endProcess()
w$closeWriter()

## Have a look at the hmp file
cat(readr::read_file("r.hmp"))

## Reset working dir
setwd(wd)
```

`loadMDP` *Load the HMDP model defined in the binary files. The model are created in memory using the external C++ library.*

Description

Load the HMDP model defined in the binary files. The model are created in memory using the external C++ library.

Usage

```
loadMDP(
  prefix = "",
 binNames = c("stateIdx.bin", "stateIdxLbl.bin", "actionIdx.bin", "actionIdxLbl.bin",
   "actionWeight.bin", "actionWeightLbl.bin", "transProb.bin", "externalProcesses.bin",
    "transWeight.bin", "transWeightLbl.bin"),
  eps = 1e-05,
  check = TRUE,
  verbose = FALSE,
  getLog = TRUE
)
```

Arguments

`prefix`	A character string with the prefix added to `binNames`. Used to identify a specific model.
`binNames`	A character vector of length 7 giving the names of the binary files storing the model.
`eps`	The sum of the transition probabilities must at most differ `eps` from one.
`check`	Check if the MDP seems correct.
`verbose`	More output when running algorithms.
`getLog`	Output the log messages.

Value

A list containing relevant information about the model such as model file names (`binNames`), time horizon (`timeHorizon`), number of states (`states`), number of states at last stage of the founder process (`founderStatesLast`), number of actions (`actions`), number of levels (`levels`), names of the weights associated to each action (`weightNames`) and a pointer `ptr` to the model object in memory. Note for models with an infinite time-horizon the states at the founder level is repeated at stage two so have something aka a double array representation.

Examples

```
## Set working dir
wd <- setwd(tempdir())

# Create the small machine repleacement problem used as an example in L.R. Nielsen and A.R.
# Kristensen. Finding the K best policies in a finite-horizon Markov decision process. European
# Journal of Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011.

## Create the MDP using a dummy replacement node
prefix<-"machine1_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
```

```
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
      w$state(label="replaced")       # v=(3,3)
         w$action(label="Dummy", weights=0, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good", end=TRUE)        # v=(4,0)
      w$state(label="average", end=TRUE)     # v=(4,1)
      w$state(label="not working", end=TRUE) # v=(4,2)
      w$state(label="replaced", end=TRUE)    # v=(4,3)
   w$endStage()
w$endProcess()
w$closeWriter()

## Load the model into memory
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
scrapValues<-c(30,10,5,0)   # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, w, termValues=scrapValues)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)



# The example given in L.R. Nielsen and A.R. Kristensen. Finding the K best
# policies in a finite-horizon Markov decision process. European Journal of
```

```
# Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011,
# does actually not have any dummy replacement node as in the MDP above. The same
# model can be created using a single dummy node at the end of the process.

## Create the MDP using a single dummy node
prefix<-"machine2_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
      w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE) # transition to sId=12 (Dummy)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
```

```
      w$state(label="good")        # v=(4,0)
         w$action(label="rep", weights=30, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="average")     # v=(4,1)
         w$action(label="rep", weights=10, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="not working") # v=(4,2)
         w$action(label="rep", weights=5, prob=c(1,0,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=5
      w$state(label="Dummy", end=TRUE)        # v=(5,0)
   w$endStage()
w$endProcess()
w$closeWriter()

## Have a look at the state-expanded hypergraph
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
runValueIte(mdp, w, termValues = 0)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)


## Reset working dir
setwd(wd)
```

`memoryMDPWriter` *Function for building an HMDP model directly in memory.*

Description

`memoryMDPWriter()` defines the same main sub-functions as `binaryMDPWriter()`, but stores states and actions directly in C++ memory instead of writing intermediate binary files. `closeWriter()` compiles the model and returns the loaded `"HMDP"` object.

Usage

```
memoryMDPWriter(
  prefix = "",
  eps = 1e-05,
  check = TRUE,
  verbose = FALSE,
  getLog = TRUE
)
```

Arguments

prefix	A character string kept for compatibility and stored in the returned object metadata.
eps	The sum of transition probabilities must at most differ eps from one when check = TRUE.
check	Check if the MDP seems correct before returning it.
verbose	More output when compiling and running algorithms.
getLog	Output the log messages.

Details

External or included processes are not supported by `memoryMDPWriter()`.

Value

A list of functions. Calling `closeWriter()` returns an "HMDP" object.

Note

Note all indexes are starting from zero (C/C++ style).

Examples

```
## Use temp dir
wd <- setwd(tempdir())

# Create a small HMDP with two levels
w<-memoryMDPWriter()
w$setWeights(c("Duration","Net reward","Items"))
w$process()
   w$stage()
      w$state(label="M0")
         w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
            w$process()
               w$stage()
                  w$state(label="D")
                     w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
```

```
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
            w$endAction()
            w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
            w$endAction()
         w$endState()
         w$state(label="C1")
            w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
            w$endAction()
            w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
            w$endAction()
         w$endState()
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
            w$endAction()
         w$endState()
         w$state(label="C1")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
            w$endAction()
         w$endState()
      w$endStage()
   w$endProcess()
w$endAction()
w$action(label="A1",weights=c(0,0,0),prob=c(2,0,1))
   w$process()
      w$stage()
         w$state(label="D")
            w$action(label="A0",weights=c(0,0,1),prob=c(1,0,1))
            w$endAction()
         w$endState()
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
            w$endAction()
            w$action(label="A1",weights=c(1,2,1),prob=c(1,0,0.5,1,1,0.5))
            w$endAction()
         w$endState()
      w$endStage()
      w$stage()
         w$state(label="C0")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
            w$endAction()
         w$endState()
         w$state(label="C1")
            w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
            w$endAction()
            w$action(label="A1",weights=c(0,10,5),prob=c(0,0,0.5,0,1,0.5))
            w$endAction()
```

```
                  w$endState()
               w$endStage()
            w$endProcess()
         w$endAction()
      w$endState()
      w$state(label="M1")
         w$action(label="A0",weights=c(0,0,0),prob=c(2,0,1))
            w$process()
               w$stage()
                  w$state(label="D")
                     w$action(label="A0",weights=c(0,0,1),prob=c(1,0,0.5,1,1,0.5))
                     w$endAction()
                  w$endState()
               w$endStage()
               w$stage()
                  w$state(label="C0")
                     w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                     w$endAction()
                  w$endState()
                  w$state(label="C1")
                     w$action(label="A0",weights=c(0,0,0),prob=c(1,0,1))
                     w$endAction()
                  w$endState()
               w$endStage()
               w$stage()
                  w$state(label="C0")
                     w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                     w$endAction()
                  w$endState()
                  w$state(label="C1")
                     w$action(label="A0",weights=c(1,4,0),prob=c(0,0,1))
                     w$endAction()
                  w$endState()
               w$endStage()
            w$endProcess()
         w$endAction()
      w$endState()
   w$endStage()
w$endProcess()
w$closeWriter()

## Info about the binary files (don't have to load the model first)
if (FALSE) {
   getBinInfoStates()
   getBinInfoActions()
}

## reset working dir
setwd(wd)
```

---

plot.HMDP *Plot the state-expanded hypergraph of the MDP.*

Description

Plot the state-expanded hypergraph of the MDP.

Usage

```
## S3 method for class 'HMDP'
plot(x, ...)
```

Arguments

x	The MDP model.
...	Arguments passed to plotHypergraph().

Value

No return value (NULL invisible), called for side effects (plotting).

See Also

getHypergraph() and plotHypergraph() for possible arguments.

Examples

```
## Set working dir
wd <- setwd(system.file("models", package = "MDP2"))

#### A finite-horizon replacement problem ####
mdp<-loadMDP("machine1_")
plot(mdp)
plot(mdp, actionColor = "label")  # colors based on labels
plot(mdp, transLabels = "state")  # label transitions with target state labels
plot(mdp, transLabels = "prob")  # label transitions with transition probabilities
plot(mdp, actionColor = "label", stateLabel = "sId|label")  # state labels are 'sId | label'
plot(mdp, stateLabel = "sIdx|label", radx = 0.01)  # adjust radx in states
plot(mdp, stateLabel = "label", actionWLabel = "none", actionLabel = "label",
     transLabels = "sId", radx = 0.01)

scrapValues <- c(30, 10, 5, 0)  # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, "Net reward" , termValues = scrapValues)
plot(mdp, actionColor = "policy")  # highlight optimal policy
plot(mdp, actionsVisible = "policy", stateLabel = "weight")  # show only optimal policy


#### An infinite-horizon maintenance problem ####
mdp<-loadMDP("hct611-1_")
plot(mdp)  # plot the first two stages
plot(mdp, actionColor = "label")  # colors based on labels
plot(mdp, actionColor = "label", stateLabel = "sId|label")  # state labels are 'sId | label'
runPolicyIteAve(mdp,"Net reward","Duration")
plot(mdp, actionColor = "policy")  # highlight optimal policy
plot(mdp, actionsVisible = "policy")  # show only optimal policy
```

```
#### An infinite-horizon hierarchical replacement problem ####
library(magrittr)
mdp<-loadMDP("cow_")
hgf <- getHypergraph(mdp)
# modify labels
dat <- hgf$nodes %>%
   dplyr::mutate(label = dplyr::case_when(
      label == "Low yield" ~ "L",
      label == "Avg yield" ~ "A",
      label == "High yield" ~ "H",
      label == "Dummy" ~ "D",
      label == "Bad genetic level" ~ "Bad",
      label == "Avg genetic level" ~ "Avg",
      label == "Good genetic level" ~ "Good",
      TRUE ~ "Error"
   ))
# assign nodes to grid ids
dat$gId[1:3]<-85:87
dat$gId[43:45]<-1:3
getGId<-function(process,stage,state) {
   if (process==0) start=18
   if (process==1) start=22
   if (process==2) start=26
   return(start + 14 * stage + state)
}
idx<-43
for (process in 0:2)
   for (stage in 0:4)
      for (state in 0:2) {
         if (stage==0 & state>0) break
         idx<-idx-1
         #cat(idx,process,stage,state,getGId(process,stage,state),"\n")
         dat$gId[idx]<-getGId(process,stage,state)
      }
hgf$nodes <- dat
# modify labels
dat <- hgf$hyperarcs %>%
   dplyr::mutate(label = dplyr::case_when(
      label == "Replace" ~ "R",
      label == "Keep" ~ "K",
      label == "Dummy" ~ "D",
      TRUE ~ "Error"
   ),
   col = dplyr::case_when(
      label == "R" ~ "deepskyblue3",
      label == "K" ~ "darkorange1",
      label == "D" ~ "black",
      TRUE ~ "Error"
   ),
   lwd = 0.5,
   label = ""
```

```
   )
hgf$hyperarcs <- dat
# plot hypergraph
oldpar <- par(mai = c(0, 0, 0, 0))
plotHypergraph(gridDim = c(14, 7), hgf, cex = 0.8, radx = 0.02, rady = 0.03)
par(oldpar)


## A simple finite-horizon MDP with action and transition weights
prefix <- file.path(tempdir(), "plot_transition_rewards_")
w <- binaryMDPWriter(prefix)
w$setWeights("Cost")
w$setTransWeights(c("Reward", "Disease"))
w$process()
   w$stage()
      w$state(label = "S1")
         w$action(
            label = "A1", weights = 2, id = c(1), pr = c(1),
            transWeights = c(20, 0.3), end = TRUE
         )
         w$action(
            label = "A2", weights = 1, id = c(0, 1), pr = c(0.3, 0.7),
            transWeights = c(25, 0.4, 15, 0.2), end = TRUE
         )
      w$endState()
   w$endStage()
   w$stage()
      w$state(label = "S2")
         w$action(
            label = "A3", weights = 3, id = c(0, 1, 2), pr = c(0.5, 0.3, 0.2),
            transWeights = c(0, 0.05, 12, 0.2, 30, 0.8), end = TRUE
         )
         w$action(
            label = "A4", weights = 2, id = c(1, 2), pr = c(0.6, 0.4),
            transWeights = c(22, 0.35, 27, 0.7), end = TRUE
         )
      w$endState()
      w$state(label = "S3")
         w$action(
            label = "A5", weights = 1, id = c(0, 1), pr = c(0.4, 0.6),
            transWeights = c(5, 0, 16, 0.25), end = TRUE
         )
         w$action(
            label = "A6", weights = 4, id = c(0, 1, 2), pr = c(0.1, 0.3, 0.6),
            transWeights = c(14, 0.15, 21, 0.45, 29, 1), end = TRUE
         )
      w$endState()
   w$endStage()
   w$stage()
      w$state(label = "S4", end = TRUE)
      w$state(label = "S5", end = TRUE)
      w$state(label = "S6", end = TRUE)
   w$endStage()
```

```
w$endProcess()
w$closeWriter()

mdp <- loadMDP(prefix, getLog = FALSE)
plot(mdp, actionColor = "label", transLabels = "weights", actionWLabel = "weight",
     radx = 0.005, rady = 0.01)

## Reset working dir
setwd(wd)
```

`plotHypergraph` *Plot parts of the state expanded hypergraph.*

Description

The plot is created based on a grid `(rows, cols)`. Each grid point is numbered from bottom to top and left to right (starting from 1), i.e. given grid point with coordinates `(r, c)` (where `(1,1)` is the top left corner and `(rows, cols)` is the bottom right corner) the grid id is '(c

- 1. – rows + r'. You must assign a node to the hypergraph to a grid point (see below).

Usage

```
plotHypergraph(
  hgf,
  gridDim,
  showGrid = FALSE,
  radx = 0.03,
  rady = 0.05,
  cex = 1,
  marX = 0.035,
  marY = 0.15,
  drawBorder = FALSE,
  actionOffset = 0.025,
  transLabels = "none",
  transLabelCex = 0.8 * cex,
  transLabelAdj = c(0.5, -0.6),
  stateLabel = "label",
  actionLabel = "label",
  actionWLabel = "none",
  actionColor = c("", "label", "policy"),
  actionsVisible = c("all", "policy"),
  connectedTo = NULL,
  recalcGrid = FALSE,
  mdp = NULL,
  ...
)
```

Arguments

hgf	A list with the hypergraph containing two data frames, normally found using getHypergraph(). The data frame nodes must have columns: sId (state id), gId (grid id) and label (node label). The data frame hyperarcs must have columns sId (head node), trans (a list-column of tail node ids), pr (a list-column of transition probabilities), actionWeights (a list-column of action weights), transWeights (a list-column of transition-by-weight matrices), aIdx (action index), label (action label), lwd (hyperarc line width), lty (hyperarc line type) and col (hyperarc color).
gridDim	A 2-dim vector (rows, cols) representing the size of the grid.
showGrid	If true show the grid points (good for debugging).
radx	Horizontal radius of the box.
rady	Vertical radius of the box.
cex	Relative size of text.
marX	Horizontal margin.
marY	Vertical margin.
drawBorder	If TRUE, draw a border around the plot region and report the outside and inside padding (good for debugging).
actionOffset	Distance used to separate actions with the same start and trans states. Set to 0 to draw overlapping actions.
transLabels	Transition-label mode. "none" draws no transition labels (the default); "custom" draws values from an optional transLabels list-column in hgf$hyperarcs; otherwise use a |-separated combination of "label", "sId", "prob", and "weights", for example "prob|weights". The older "state" spelling is treated as "label".
transLabelCex	Relative size of transition-label text.
transLabelAdj	Position adjustment passed to textempty() for transition labels, drawn at the middle of each split-to-transition branch.
stateLabel	What to plot in states. "custom" uses a stateLabel column in hgf$nodes; otherwise use a |-separated combination of "label" (state label, default), "sId" (state id), "sIdx" (stage-based state index), and "weight" (optimal weight of the state).
actionLabel	What to plot near the split. One of "none", "custom" (uses an actionLabel column in hgf$hyperarcs), or a |-separated combination of "label" (action label, default) and "aIdx".
actionWLabel	What to plot from the start state to the split. One of "none" (default), "weight", or "custom" (uses an actionWLabel column in hgf$hyperarcs).
actionColor	Action coloring scheme. Default "" uses black lines. "label" uses different colors based on the action labels. "policy" highlights the current policy.
actionsVisible	Action visibility mode. "all" (default) shows all actions. "policy" only shows actions in the current policy.
connectedTo	Optional vector of state ids. If supplied, plot only states reachable from these states by following visible hyperarcs forward, and trim hyperarcs and transition-level data to the remaining states.

recalcGrid	If TRUE and connectedTo is supplied, recalculate the grid for the visible nodes. Nodes keep their original columns, but visible nodes within each column are placed consecutively from the top and the number of grid rows is reduced to the maximum number of visible nodes in any column.
mdp	The MDP model. Required if stateLabel contains "weight", actionColor = "policy", or actionsVisible = "policy".
...	Graphical parameters passed to textempty.

Value

No return value (NULL invisible), called for side effects (plotting).

See Also

getHypergraph() and plot.HMDP().

Examples

```
## Set working dir
wd <- setwd(system.file("models", package = "MDP2"))

#### A finite-horizon replacement problem ####
mdp<-loadMDP("machine1_")
plot(mdp)
plot(mdp, actionColor = "label")  # colors based on labels
plot(mdp, transLabels = "state")  # label transitions with target state labels
plot(mdp, transLabels = "prob")  # label transitions with transition probabilities
plot(mdp, actionColor = "label", stateLabel = "sId|label") # state labels are 'sId | label'
plot(mdp, stateLabel = "sIdx|label", radx = 0.01)  # adjust radx in states
plot(mdp, stateLabel = "label", actionWLabel = "none", actionLabel = "label",
     transLabels = "sId", radx = 0.01)

scrapValues <- c(30, 10, 5, 0)  # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, "Net reward" , termValues = scrapValues)
plot(mdp, actionColor = "policy")  # highlight optimal policy
plot(mdp, actionsVisible = "policy", stateLabel = "weight")  # show only optimal policy


#### An infinite-horizon maintenance problem ####
mdp<-loadMDP("hct611-1_")
plot(mdp)  # plot the first two stages
plot(mdp, actionColor = "label")  # colors based on labels
plot(mdp, actionColor = "label", stateLabel = "sId|label") # state labels are 'sId | label'
runPolicyIteAve(mdp,"Net reward","Duration")
plot(mdp, actionColor = "policy")  # highlight optimal policy
plot(mdp, actionsVisible = "policy")  # show only optimal policy


#### An infinite-horizon hierarchical replacement problem ####
library(magrittr)
mdp<-loadMDP("cow_")
```

```
hgf <- getHypergraph(mdp)
# modify labels
dat <- hgf$nodes %>%
   dplyr::mutate(label = dplyr::case_when(
      label == "Low yield" ~ "L",
      label == "Avg yield" ~ "A",
      label == "High yield" ~ "H",
      label == "Dummy" ~ "D",
      label == "Bad genetic level" ~ "Bad",
      label == "Avg genetic level" ~ "Avg",
      label == "Good genetic level" ~ "Good",
      TRUE ~ "Error"
   ))
# assign nodes to grid ids
dat$gId[1:3]<-85:87
dat$gId[43:45]<-1:3
getGId<-function(process,stage,state) {
   if (process==0) start=18
   if (process==1) start=22
   if (process==2) start=26
   return(start + 14 * stage + state)
}
idx<-43
for (process in 0:2)
   for (stage in 0:4)
      for (state in 0:2) {
         if (stage==0 & state>0) break
         idx<-idx-1
         #cat(idx,process,stage,state,getGId(process,stage,state),"\n")
         dat$gId[idx]<-getGId(process,stage,state)
      }
hgf$nodes <- dat
# modify labels
dat <- hgf$hyperarcs %>%
   dplyr::mutate(label = dplyr::case_when(
      label == "Replace" ~ "R",
      label == "Keep" ~ "K",
      label == "Dummy" ~ "D",
      TRUE ~ "Error"
   ),
   col = dplyr::case_when(
      label == "R" ~ "deepskyblue3",
      label == "K" ~ "darkorange1",
      label == "D" ~ "black",
      TRUE ~ "Error"
   ),
   lwd = 0.5,
   label = ""
   )
hgf$hyperarcs <- dat
# plot hypergraph
oldpar <- par(mai = c(0, 0, 0, 0))
plotHypergraph(gridDim = c(14, 7), hgf, cex = 0.8, radx = 0.02, rady = 0.03)
```

```
par(oldpar)


## A simple finite-horizon MDP with action and transition weights
prefix <- file.path(tempdir(), "plot_transition_rewards_")
w <- binaryMDPWriter(prefix)
w$setWeights("Cost")
w$setTransWeights(c("Reward", "Disease"))
w$process()
   w$stage()
      w$state(label = "S1")
         w$action(
            label = "A1", weights = 2, id = c(1), pr = c(1),
            transWeights = c(20, 0.3), end = TRUE
         )
         w$action(
            label = "A2", weights = 1, id = c(0, 1), pr = c(0.3, 0.7),
            transWeights = c(25, 0.4, 15, 0.2), end = TRUE
         )
      w$endState()
   w$endStage()
   w$stage()
      w$state(label = "S2")
         w$action(
            label = "A3", weights = 3, id = c(0, 1, 2), pr = c(0.5, 0.3, 0.2),
            transWeights = c(0, 0.05, 12, 0.2, 30, 0.8), end = TRUE
         )
         w$action(
            label = "A4", weights = 2, id = c(1, 2), pr = c(0.6, 0.4),
            transWeights = c(22, 0.35, 27, 0.7), end = TRUE
         )
      w$endState()
      w$state(label = "S3")
         w$action(
            label = "A5", weights = 1, id = c(0, 1), pr = c(0.4, 0.6),
            transWeights = c(5, 0, 16, 0.25), end = TRUE
         )
         w$action(
            label = "A6", weights = 4, id = c(0, 1, 2), pr = c(0.1, 0.3, 0.6),
            transWeights = c(14, 0.15, 21, 0.45, 29, 1), end = TRUE
         )
      w$endState()
   w$endStage()
   w$stage()
      w$state(label = "S4", end = TRUE)
      w$state(label = "S5", end = TRUE)
      w$state(label = "S6", end = TRUE)
   w$endStage()
w$endProcess()
w$closeWriter()

mdp <- loadMDP(prefix, getLog = FALSE)
plot(mdp, actionColor = "label", transLabels = "weights", actionWLabel = "weight",
```

```
     radx = 0.005, rady = 0.01)

## Reset working dir
setwd(wd)
```

---

randomHMDP *Generate a "random" HMDP stored in a set of binary files.*

Description

Generate a "random" HMDP stored in a set of binary files.

Usage

```
randomHMDP(
  prefix = "",
  levels = 3,
  timeHorizon = c(Inf, 3, 4),
  states = c(2, 4, 5),
  actions = c(1, 2),
  childProcessPr = 0.5,
  externalProcessPr = 0,
  rewards = c(0, 100),
  durations = c(1, 10),
  rewardName = "Reward",
  durationName = "Duration"
)
```

Arguments

prefix	A character string with the prefix added to the file(s).
levels	Maximum number of levels. Set childProcessPr = 1 if want exact this number of levels.
timeHorizon	The time horizon for each level (vector). For the founder the time-horizon can be Inf.
states	Number of states at each stage at a given level (vector of length levels)
actions	Min and max number of actions at a state.
childProcessPr	Probability of creating a child process when define action.
externalProcessPr	Probability of creating an external process given that we create a child process. Only works if levels>2 and and currently does not generate external processes which include external processes.
rewards	Min and max reward used.
durations	Min and max duration used.
rewardName	Weight name used for reward.
durationName	Weight name used for duration.

Value

The file prefix (character).

`runCalcWeights` *Calculate weights based on current policy. Normally run after an optimal policy has been found.*

Description

Calculate weights based on current policy. Normally run after an optimal policy has been found.

Usage

```
runCalcWeights(
  mdp,
  wLbl,
  criterion = "expected",
  durLbl = NULL,
  rate = 0,
  rateBase = 1,
  discountFactor = NULL,
  termValues = NULL,
  discountMethod = "continuous"
)
```

Arguments

`mdp`	The MDP loaded using `loadMDP()`.
`wLbl`	The label of the weight we consider.
`criterion`	The Bellman operator shortcut. If `expected` use expected weights, if `discount` use discounted expected weights, if `average` use average expected weights, if `min` use minimum-successor weights, if `max` use maximum-successor weights, if `secondMoment` use the second moment of total accumulated weight, and if `variance` use the law-of-total-variance recursion under the current policy.
`durLbl`	The label of the duration/time such that discount rates can be calculated.
`rate`	The interest rate.
`rateBase`	The time-horizon the rate is valid over.
`discountFactor`	The discount rate for one time unit. If specified `rate` and `rateBase` are not used to calculate the discount rate.
`termValues`	The terminal values used (values of the last stage in the MDP).
`discountMethod`	Either 'continuous' or 'discrete', corresponding to discount factor `exp(-rate/rateBase)` or `1/(1 + rate/rateBase)`, respectively. Only used if `discountFactor` is `NULL`.

Value

Nothing.

Examples

```
## Set working dir
wd <- setwd(tempdir())

# Create the small machine repleacement problem used as an example in L.R. Nielsen and A.R.
# Kristensen. Finding the K best policies in a finite-horizon Markov decision process. European
# Journal of Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011.

## Create the MDP using a dummy replacement node
prefix<-"machine1_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
```

```
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
      w$state(label="replaced")       # v=(3,3)
         w$action(label="Dummy", weights=0, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good", end=TRUE)        # v=(4,0)
      w$state(label="average", end=TRUE)     # v=(4,1)
      w$state(label="not working", end=TRUE) # v=(4,2)
      w$state(label="replaced", end=TRUE)    # v=(4,3)
   w$endStage()
w$endProcess()
w$closeWriter()

## Load the model into memory
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
scrapValues<-c(30,10,5,0)   # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, w, termValues=scrapValues)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)



# The example given in L.R. Nielsen and A.R. Kristensen. Finding the K best
# policies in a finite-horizon Markov decision process. European Journal of
# Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011,
# does actually not have any dummy replacement node as in the MDP above. The same
# model can be created using a single dummy node at the end of the process.

## Create the MDP using a single dummy node
```

```
prefix<-"machine2_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
      w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE) # transition to sId=12 (Dummy)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good")        # v=(4,0)
         w$action(label="rep", weights=30, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="average")     # v=(4,1)
         w$action(label="rep", weights=10, prob=c(1,0,1), end=TRUE)
```

```
      w$endState()
      w$state(label="not working") # v=(4,2)
         w$action(label="rep", weights=5, prob=c(1,0,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=5
      w$state(label="Dummy", end=TRUE)        # v=(5,0)
   w$endStage()
w$endProcess()
w$closeWriter()

## Have a look at the state-expanded hypergraph
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
runValueIte(mdp, w, termValues = 0)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)


## Reset working dir
setwd(wd)
```

`runPolicyIteAve` *Perform policy iteration using the average expected-weight Bellman operator on the MDP.*

Description

The policy can afterwards be received using functions `getPolicy` and `getPolicyW`.

Usage

```
runPolicyIteAve(
  mdp,
```

```
  w,
  dur,
  maxIte = 100,
  objective = c("max", "min"),
  getLog = TRUE
)
```

Arguments

mdp	The MDP loaded using loadMDP().
w	The label of the weight we optimize.
dur	The label of the duration/time such that discount rates can be calculated.
maxIte	Max number of iterations. If the model does not satisfy the unichain assumption the algorithm may loop.
objective	Optimize by maximizing ("max") or minimizing ("min") the Bellman value.
getLog	Output the log messages.

Value

The optimal gain (g) calculated.

See Also

getPolicy().

---

runPolicyIteDiscount *Perform policy iteration using the discounted expected-weight Bellman operator on the MDP.*

Description

The policy can afterwards be received using functions getPolicy and getPolicyW.

Usage

```
runPolicyIteDiscount(
  mdp,
  w,
  dur,
  rate = 0,
  rateBase = 1,
  discountFactor = NULL,
  maxIte = 100,
  discountMethod = "continuous",
  objective = c("max", "min"),
  getLog = TRUE
)
```

Arguments

mdp	The MDP loaded using loadMDP().
w	The label of the weight we optimize.
dur	The label of the duration/time such that discount rates can be calculated.
rate	The interest rate.
rateBase	The time-horizon the rate is valid over.
discountFactor	The discount rate for one time unit. If specified rate and rateBase are not used to calculate the discount rate.
maxIte	Max number of iterations. If the model does not satisfy the unichain assumption the algorithm may loop.
discountMethod	Either 'continuous' or 'discrete', corresponding to discount factor exp(-rate/rateBase) or 1/(1 + rate/rateBase), respectively. Only used if discountFactor is NULL.
objective	Optimize by maximizing ("max") or minimizing ("min") the Bellman value.
getLog	Output the log messages.

Value

Nothing.

See Also

getPolicy().

---

runValueIte *Perform value iteration on the MDP.*

Description

If the MDP has a finite time-horizon then arguments times and eps are ignored.

Usage

```
runValueIte(
  mdp,
  w,
  dur = NULL,
  rate = 0,
  rateBase = 1,
  discountFactor = NULL,
  maxIte = 100,
  eps = 1e-05,
  termValues = NULL,
  g = NULL,
  objective = c("max", "min"),
```

```
 bellmanOp = c("auto", "expected", "discount", "average", "min", "max", "secondMoment"),
  getLog = TRUE,
  discountMethod = "continuous"
)
```

Arguments

mdp	The MDP loaded using loadMDP().
w	The label of the weight we optimize.
dur	The label of the duration/time such that discount rates can be calculated.
rate	Interest rate.
rateBase	The time-horizon the rate is valid over.
discountFactor	The discount rate for one time unit. If specified rate and rateBase are not used to calculate the discount rate.
maxIte	The max number of iterations value iteration is performed.
eps	Stopping tolerance. If $max(w(t)-w(t+1)) < eps$ then stop the algorithm, i.e the policy becomes epsilon optimal (see Puterman p161).
termValues	The terminal values used (values of the last stage in the MDP).
g	Average weight. If specified then do a single iteration using the update equations under the average expected-weight Bellman operator with the specified g value.
objective	Optimize by maximizing ("max") or minimizing ("min") the Bellman value.
bellmanOp	Bellman operator. Use "auto" for existing behavior, "min" for the minimum-successor operator, "max" for the maximum-successor operator, or "secondMoment" for the second moment of total accumulated weight.
getLog	Output the log messages.
discountMethod	Either 'continuous' or 'discrete', corresponding to discount factor exp(-rate/rateBase) or 1/(1 + rate/rateBase), respectively. Only used if discountFactor is NULL.

Value

NULL (invisible)

References

Puterman, M. Markov Decision Processes, Wiley-Interscience, 1994.

Examples

```
## Set working dir
wd <- setwd(tempdir())

# Create the small machine repleacement problem used as an example in L.R. Nielsen and A.R.
# Kristensen. Finding the K best policies in a finite-horizon Markov decision process. European
# Journal of Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011.

## Create the MDP using a dummy replacement node
```

```
prefix<-"machine1_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
      w$state(label="replaced")       # v=(3,3)
         w$action(label="Dummy", weights=0, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good", end=TRUE)        # v=(4,0)
      w$state(label="average", end=TRUE)     # v=(4,1)
```

```
      w$state(label="not working", end=TRUE) # v=(4,2)
      w$state(label="replaced", end=TRUE)    # v=(4,3)
   w$endStage()
w$endProcess()
w$closeWriter()

## Load the model into memory
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
scrapValues<-c(30,10,5,0)   # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, w, termValues=scrapValues)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)



# The example given in L.R. Nielsen and A.R. Kristensen. Finding the K best
# policies in a finite-horizon Markov decision process. European Journal of
# Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011,
# does actually not have any dummy replacement node as in the MDP above. The same
# model can be created using a single dummy node at the end of the process.

## Create the MDP using a single dummy node
prefix<-"machine2_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
```

```
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
      w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE) # transition to sId=12 (Dummy)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good")        # v=(4,0)
         w$action(label="rep", weights=30, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="average")     # v=(4,1)
         w$action(label="rep", weights=10, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="not working") # v=(4,2)
         w$action(label="rep", weights=5, prob=c(1,0,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=5
      w$state(label="Dummy", end=TRUE)        # v=(5,0)
   w$endStage()
w$endProcess()
w$closeWriter()

## Have a look at the state-expanded hypergraph
mdp<-loadMDP(prefix)
mdp
```

```
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
runValueIte(mdp, w, termValues = 0)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)


## Reset working dir
setwd(wd)
```

---

saveMDP *Save the MDP to binary files*

Description

Currently do not save external files.

Usage

```
saveMDP(mdp, prefix = "", getLog = TRUE)
```

Arguments

mdp	The MDP loaded using loadMDP().
prefix	A character string with the prefix added to binNames. Used to identify a specific model.
getLog	Output the log as a message.

Value

???

`setPolicy` *Modify the current policy by setting policy action of states.*

Description

If the policy does not contain all states then the actions from the previous optimal policy are used.

Usage

```
setPolicy(mdp, policy)
```

Arguments

`mdp`	The MDP loaded using `loadMDP()`.
`policy`	A data frame with two columns state id `sId` and action index `aIdx`.

Value

NULL (invisible)

Examples

```
## Set working dir
wd <- setwd(tempdir())

# Create the small machine repleacement problem used as an example in L.R. Nielsen and A.R.
# Kristensen. Finding the K best policies in a finite-horizon Markov decision process. European
# Journal of Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011.

## Create the MDP using a dummy replacement node
prefix<-"machine1_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
```

```
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(1,3,1), end=TRUE)
      w$endState()
      w$state(label="replaced")       # v=(3,3)
         w$action(label="Dummy", weights=0, prob=c(1,3,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good", end=TRUE)        # v=(4,0)
      w$state(label="average", end=TRUE)     # v=(4,1)
      w$state(label="not working", end=TRUE) # v=(4,2)
      w$state(label="replaced", end=TRUE)    # v=(4,3)
   w$endStage()
w$endProcess()
w$closeWriter()

## Load the model into memory
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
scrapValues<-c(30,10,5,0)   # scrap values (the values of the 4 states at stage 4)
runValueIte(mdp, w, termValues=scrapValues)
```

```
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)



# The example given in L.R. Nielsen and A.R. Kristensen. Finding the K best
# policies in a finite-horizon Markov decision process. European Journal of
# Operational Research, 175(2):1164-1179, 2006. doi:10.1016/j.ejor.2005.06.011,
# does actually not have any dummy replacement node as in the MDP above. The same
# model can be created using a single dummy node at the end of the process.

## Create the MDP using a single dummy node
prefix<-"machine2_"
w <- binaryMDPWriter(prefix)
w$setWeights(c("Net reward"))
w$process()
   w$stage()   # stage n=0
      w$state(label="Dummy")          # v=(0,0)
         w$action(label="buy", weights=-100, prob=c(1,0,0.7, 1,1,0.3), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=1
      w$state(label="good")           # v=(1,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.6, 1,1,0.4), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(1,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.6, 1,2,0.4), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=2
      w$state(label="good")           # v=(2,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.5, 1,1,0.5), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(2,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.5, 1,2,0.5), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(2,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
      w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE) # transition to sId=12 (Dummy)
      w$endState()
   w$endStage()
```

```
   w$stage()   # stage n=3
      w$state(label="good")           # v=(3,0)
         w$action(label="mt", weights=55, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=70, prob=c(1,0,0.2, 1,1,0.8), end=TRUE)
      w$endState()
      w$state(label="average")        # v=(3,1)
         w$action(label="mt", weights=40, prob=c(1,0,1), end=TRUE)
         w$action(label="nmt", weights=50, prob=c(1,1,0.2, 1,2,0.8), end=TRUE)
      w$endState()
      w$state(label="not working")    # v=(3,2)
         w$action(label="mt", weights=30, prob=c(1,0,1), end=TRUE)
         w$action(label="rep", weights=5, prob=c(3,12,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=4
      w$state(label="good")        # v=(4,0)
         w$action(label="rep", weights=30, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="average")     # v=(4,1)
         w$action(label="rep", weights=10, prob=c(1,0,1), end=TRUE)
      w$endState()
      w$state(label="not working") # v=(4,2)
         w$action(label="rep", weights=5, prob=c(1,0,1), end=TRUE)
      w$endState()
   w$endStage()
   w$stage()   # stage n=5
      w$state(label="Dummy", end=TRUE)        # v=(5,0)
   w$endStage()
w$endProcess()
w$closeWriter()

## Have a look at the state-expanded hypergraph
mdp<-loadMDP(prefix)
mdp
plot(mdp)

getInfo(mdp, withList = FALSE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = TRUE)
getInfo(mdp, withList = FALSE, dfLevel = "action", asStringsActions = FALSE)

## Perform value iteration
w<-"Net reward"             # label of the weight we want to optimize
runValueIte(mdp, w, termValues = 0)
getPolicy(mdp)     # optimal policy

## Calculate the weights of the policy always to maintain
library(magrittr)
policy <- getInfo(mdp, withList = FALSE, dfLevel = "action")$df %>%
   dplyr::filter(label_action == "mt") %>%
   dplyr::select(sId, aIdx)
setPolicy(mdp, policy)
runCalcWeights(mdp, w, termValues=scrapValues)
getPolicy(mdp)
```

```
## Reset working dir
setwd(wd)
```

Index