Package 'multiobjectiveMDP'

March 6, 2026

Title Solution Methods for Multi-Objective Markov Decision Processes

Version 1.0.0

Description Compendium of the most representative algorithms in print---vector-valued dynamic programming, linear programming, policy iteration, the weighting factor approach---for solving multi-objective Markov decision processes, with or without reward discount, over a finite or infinite horizon. Mifrani, A. (2024) <doi:10.1007/s10479-024-06439-x>; Mifrani, A. & Noll, D. <doi:10.48550/arXiv.2502.13697>; Wakuta, K. (1995) <doi:10.1016/0304-4149(94)00064-Z>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.2

Imports dplyr, linprog, lintools, nsga2R, pracma, prodlim, stats

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Anas Mifrani [aut, cre, cph] (ORCID: <https://orcid.org/0009-0005-1373-9028>)

Maintainer Anas Mifrani <anas.mifrani@math.univ-toulouse.fr>

Repository CRAN

Date/Publication 2026-03-06 18:40:02 UTC

Contents

`are_valid_finite_horizon_rewards` *Determine whether a numeric list represents a valid reward structure for finite-horizon problems*

Description

`are_valid_finite_horizon_rewards()` returns `TRUE` if `rewards` constitutes a valid reward structure for a finite-horizon problem. In the opposite case, it returns `FALSE` together with an explanatory message.

Usage

```
are_valid_finite_horizon_rewards(rewards)
```

Arguments

`rewards`	A numeric list.

Details

This function juxtaposes `rewards` against the template described in the documentation of `evaluate_finite_horizon_MMDP_` for finite-horizon reward lists.

Value

A boolean: `TRUE` if `rewards` defines a valid finite-horizon reward structure, `FALSE` otherwise.

See Also

evaluate_finite_horizon_MMDP_markov_policy() for the template of a finite-horizon reward list.

Examples

```
set.seed(1234)
MMDP <- generate_rand_MMDP(2, list(c(1, 2), c(1, 2, 3)), horizon = 5, no_objectives = 2)
rewards <- MMDP$R
are_valid_finite_horizon_rewards(rewards)
```

---

are_valid_finite_horizon_transition_probabilities

Determine whether a numeric list represents a valid transition probability structure for finite-horizon problems

---

Description

are_valid_finite_horizon_transition_probabilities() returns TRUE if transition_probabilities constitutes a valid transition probability structure for a finite-horizon problem. In the opposite case, it returns FALSE in addition to an explanation.

Usage

```
are_valid_finite_horizon_transition_probabilities(transition_probabilities)
```

Arguments

transition_probabilities
: A numeric list.

Details

This function juxtaposes transition_probabilities against the template described in the documentation of evaluate_finite_horizon_MMDP_markov_policy() for finite-horizon transition probability lists.

Value

A boolean: TRUE if transition_probabilities defines a valid finite-horizon transition probability structure, FALSE otherwise.

See Also

evaluate_finite_horizon_MMDP_markov_policy()

Examples

```
set.seed(1234)
MMDP <- generate_rand_MMDP(2, list(c(1, 2), c(1, 2, 3)), horizon = 5, no_objectives = 2)
transition_probabilities <- MMDP$P
are_valid_finite_horizon_transition_probabilities(transition_probabilities)
```

---

are_valid_infinite_horizon_rewards

Determine whether a numeric list represents a valid reward structure for infinite-horizon problems

---

Description

`are_valid_infinite_horizon_rewards()` returns `TRUE` if `stationary_rewards` constitutes a valid reward structure for a stationary infinite-horizon problem. Otherwise, it returns `FALSE` in addition to an explanation.

Usage

```
are_valid_infinite_horizon_rewards(stationary_rewards)
```

Arguments

`stationary_rewards`
: A numeric list.

Details

This function juxtaposes `stationary_rewards` against the template described in the documentation of `evaluate_discounted_MMDP_pure_policy()` for infinite-horizon reward lists.

Value

A boolean: `TRUE` if `stationary_rewards` defines a valid infinite-horizon reward structure, `FALSE` otherwise.

See Also

`evaluate_discounted_MMDP_pure_policy()`

Examples

```
set.seed(1234)
stationary_MMDP <- generate_rand_MMDP(2, list(c(1, 2), c(1, 2, 3)),
                                      horizon = Inf, no_objectives = 2)
stationary_rewards <- stationary_MMDP$R
are_valid_infinite_horizon_rewards(stationary_rewards)
```

`are_valid_infinite_horizon_transition_probabilities` *Determine whether a numeric list represents a valid transition probability structure for infinite-horizon problems*

Description

`are_valid_infinite_horizon_transition_probabilities()` returns `TRUE` if `stationary_rewards` constitutes a valid transition probability structure for a stationary infinite-horizon problem. In the opposite case, it returns `FALSE` in addition to an explanation.

Usage

```
are_valid_infinite_horizon_transition_probabilities(
  stationary_transition_probabilities
)
```

Arguments

`stationary_transition_probabilities`
: A numeric list.

Details

This function juxtaposes `stationary_transition_probabilities` against the template described in the documentation of `evaluate_discounted_MMDP_pure_policy()` for infinite-horizon transition probability lists.

Value

A boolean: `TRUE` if `stationary_transition_probabilities` defines a valid infinite-horizon transition probability structure, `FALSE` otherwise.

See Also

`evaluate_discounted_MMDP_pure_policy()`

Examples

```
set.seed(1234)
stationary_MMDP <- generate_rand_MMDP(2, list(c(1, 2), c(1, 2, 3)),
                                      horizon = Inf, no_objectives = 2)
stationary_transition_probabilities <- stationary_MMDP$P
are_valid_infinite_horizon_transition_probabilities(stationary_transition_probabilities)
```

compromise_solution *Calculate the compromise solution among a set of objective vectors*

Description

`compromise_solution()` determines which of a finite set `value_set` of objective vectors is geometrically the closest to the ideal point that maximizes all components simultaneously.

Usage

```
compromise_solution(value_set, p = 2)
```

Arguments

`value_set`	A numeric matrix whose columns contain the objective vectors.
`p`	Either (1) an integer greater than or equal to 1 or (2) `Inf`. This argument defines the norm with respect to which distances are evaluated. The default value `2` yields the Euclidean norm.

Value

A numeric vector giving the column of `value_set` that minimizes the l_p distance to the ideal point.

References

Yu, P. L. (2013). *Multiple-criteria decision making: concepts, techniques, and extensions* (Vol. 30). Springer Science & Business Media.

Examples

```
# Construct a set of three points: (3, 1), (2, 2) and (8, -1/2). The ideal point here is (8, 2).
value_set <- matrix(c(3, 1, 2, 2, 8, -1/2), nrow = 2)
# Which of the three points is closest to the ideal under, say, the Euclidean norm?
compromise_solution(value_set)
# What of the sup norm?
compromise_solution(value_set, Inf)
```

discounted_bellman_operator *Apply a stationary Bellman-type operator to a vector-valued value function*

Description

`discounted_bellman_operator()` is an adjunct to `solve_discounted_MMDP_policy_iteration()` and `solve_discounted_MDP_policy_iteration()` that operates a standard linear transformation on a value function `value_function` for a stationary infinite-horizon multi-objective Markov decision process.

Usage

```
discounted_bellman_operator(
  stationary_transition_probabilities,
  stationary_rewards,
  stationary_policy,
  value_function,
  rho
)
```

Arguments

`stationary_transition_probabilities`	A numeric list defining the (stationary) transition probabilities of the infinite-horizon model under study. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.
`stationary_rewards`	A numeric list defining the (stationary) reward structure. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.
`stationary_policy`	An integer vector, of length equal to the number of states, representing a stationary policy. Each element in the vector gives the index of the action prescribed for the corresponding state.
`value_function`	A numeric matrix, with one column per state, representing the value function to be transformed. The `s`-th column is the value vector corresponding to initial state `s`.
`rho`	A number greater than or equal to 0 but strictly less than 1 to use as a reward discount factor.

Details

This function uses `value_function` and `stationary_policy` to compute a new value function. Let $R_\pi(s)$ denote the reward vector for executing policy `stationary_policy` in state s— which is to say `stationary_rewards[[1]][[s]][[stationary_policy[s]]]`—and let R_π denote the matrix formed by these column vectors. Similarly, let P_π be the square matrix whose (s, j)-th entry is given by the probability of transition from state s to state j under `stationary_policy`, i.e., `stationary_probabilities[[1]][[s]][[stationary_policy[s]]][[j]]`. Then, if the letter V denotes `value_function`, this function returns the quantity

$$R_\pi + \rho V P'_\pi,$$

where P'_π denotes the transpose of P_π. The resulting matrix bears the same dimensions as `value_function`.

In the special case where `value_function` stores the value function of `stationary_policy`, the transformation leaves `value_function` unchanged as per standard discounted MDP theory.

Value

A numeric matrix having the dimensions of `value_function`, representing the new value function.

References

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

See Also

`solve_discounted_MDP_policy_iteration()` for an implementation of the standard policy iteration algorithm and `solve_discounted_MMDP_policy_iteration()` for its multi-objective extension.

To ensure that the transition probability and reward lists passed to the function are acceptable, use `are_valid_infinite_horizon_transition_probabilities()` and `are_valid_infinite_horizon_rewards()`.

Examples

```
# Generate a discounted infinite-horizon bi-objective MDP with two states
# and action sets that provide two actions for state 1 and three for state 2
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2, 3))
no_objectives <- 2
stationary_MMDP <- generate_rand_MMDP(no_states, action_sets, horizon = Inf, no_objectives)
stationary_transition_probabilities <- stationary_MMDP$P
stationary_rewards <- stationary_MMDP$R


# Consider the stationary policy that prescribes action 1 for state 1 and action 3 for state 2
stationary_policy <- c(1, 3)
# Evaluate this policy assuming a reward discount factor of .7, display the value function
rho <- .7
value_function <- evaluate_discounted_MMDP_pure_policy(stationary_transition_probabilities,
                                            stationary_rewards, stationary_policy, rho)
value_function
# What happens when we subject a policy's value function
# to the transformation defined by the policy itself?
transformed_value_function <- discounted_bellman_operator(stationary_transition_probabilities,
                                                          stationary_rewards,
                                                          stationary_policy,
                                                          value_function, rho)
transformed_value_function
```

efficient_subset_sort_prune

Find the Pareto efficient subset of a set of vectors

Description

`efficient_subset_sort_prune()` enumerates the efficient column vectors of a matrix according to the componentwise order that prefers larger components to smaller ones.

Usage

```
efficient_subset_sort_prune(X)
```

Arguments

`X`	A numeric matrix.

Value

A list comprised of:

- A numeric matrix formed by the efficient columns of `X`, and
- the indices of those columns in the original matrix.

If `X`'s columns are incomparable, the function will return `X`.

Examples

```
# The points in the following set are (1, 2), (0, 0) and (3, -6)
X <- matrix(c(1, 2, 0, 0, 3, -6), nrow = 2, ncol = 3)
# (1, 2) and (3, -6) are efficient whereas (0, 0) is dominated by (1, 2)
efficient_subset_sort_prune(X)
```

evaluate_discounted_MMDP_pure_policy

Evaluate a stationary policy in a discounted infinite-horizon multi-objective Markov decision process

Description

`evaluate_discounted_MMDP_pure_policy()` calculates the expected discounted reward of a stationary deterministic policy in an infinite-horizon multi-objective Markov decision process.

Usage

```
evaluate_discounted_MMDP_pure_policy(
  stationary_transition_probabilities,
  stationary_rewards,
  stationary_policy,
  rho
)
```

Arguments

`stationary_transition_probabilities`
: A numeric list describing the (stationary) transition probabilities of the infinite-horizon model in which `stationary_policy` is to be evaluated. The list contains a single sublist with one component per state that breaks down as follows:

  - For each state `s`, `stationary_transition_probabilities[[1]][[s]]` should be a numeric list whose length equals the number of actions permissible in `s`.
  - For each state `s` and each action `a`, the entry `stationary_transition_probabilities[[1]][[s]][`

`stationary_rewards`
: A numeric list describing the model's (stationary) reward structure. The list contains a single sublist with one component per state that breaks down as follows:

  - For each state `s` we expect `stationary_rewards[[1]][[s]]` to be a numeric list whose length equals the number of actions permissible in `s`.
  - Each third-level entry `stationary_rewards[[1]][[s]][[a]]` should be a numeric vector that gives the reward for selecting action `a` in state `s` at epoch `t`.

`stationary_policy`
: An integer vector, of length equal to the number of states as determined by `stationary_transition_probabilities` and `stationary_rewards`, indicating the action prescribed by the policy for each state.

`rho`
: A number greater than or equal to 0 but strictly less than 1 to use as a reward discount factor.

Details

Let S denote the number of states. Let π be a (Markov) stationary deterministic policy and let $v^\pi(s) = (v_1^\pi(s), \ldots, v_k^\pi(s))$ denote the expected discounted reward vector it achieves over an infinite horizon assuming the initial state is s. For each component v_i^π of this value function, Puterman (2014) shows that

$$v_i^\pi = (I - \rho P_\pi)^{-1} R_{\pi,i},$$

where v_i^π is viewed as an S-dimensional vector having entries $v_i^\pi(s)$; P_π is the transition probability matrix induced by π; $R_{\pi,i}$ is the S-dimensional vector given by the i-th reward of executing policy π in each state s; and I is the identity matrix.

Value

A numeric matrix, one row per objective and one column per state, in which the s-th column contains the value vector achieved from initial state s.

Examples

```
set.seed(1234)
stationary_MMDP <- generate_rand_MMDP(no_states = 2, action_sets = list(c(1, 2), c(1, 2, 3)),
                                      horizon = Inf, no_objectives = 2)
stationary_transition_probabilities <- stationary_MMDP$P
stationary_rewards <- stationary_MMDP$R
rho <- .7
# What's the value of the stationary policy that dictates action 2 in state 1
# and action 3 in state 2?
stationary_policy <- c(2, 3)
evaluate_discounted_MMDP_pure_policy(stationary_transition_probabilities,
                                     stationary_rewards, stationary_policy, rho)
```

---

evaluate_finite_horizon_MMDP_markov_policy

Evaluate a Markov deterministic policy for a finite-horizon multi-objective Markov decision process

---

Description

`evaluate_finite_horizon_MMDP_markov_policy()` recursively calculates the value function of a Markov deterministic policy for a finite-horizon multi-objective Markov decision process.

Usage

```
evaluate_finite_horizon_MMDP_markov_policy(
  transition_probabilities,
  rewards,
  policy,
  rho = 1
)
```

Arguments

`transition_probabilities`
: A numeric list containing the transition probabilities of the finite-horizon model in which `policy` is being evaluated. The length of the list is the number of decision epochs (excluding the terminal epoch). There should be three levels in this list:

  - For each decision epoch `t`, `transition_probabilities[[t]]` is expected to be a numeric list whose length equals the number of states in the model.
  - For each state `s`, `transition_probabilities[[t]][[s]]` should be a numeric list whose length equals the number of actions permissible in `s`.

- Each third-level entry `transition_probabilities[[t]][[s]][[a]]` should be a numeric vector that gives the probability distribution of the states at epoch `t+1` if the state at epoch `t` was `s` and the action selected was `a`.

`rewards` A numeric list that defines the rewards of the finite-horizon model in which `policy` is being evaluated. The length of the list is the number of epochs in the model (including the terminal epoch). There should be three levels in this list:

- For all epochs `t`, `rewards[[t]]` is expected to be a numeric list whose length equals the number of states in the model.
- If `t` is a decision epoch, then for each state `s` we expect `rewards[[t]][[s]]` to be a numeric list whose length equals the number of actions permissible in `s`. If `t` is terminal, then `rewards[[t]][[s]]` should be a numeric vector that gives the reward for the MMDP terminating in state `s`.
- Given a decision epoch `t` and a state `s`, each third-level entry `rewards[[t]][[s]][[a]]` should be a numeric vector that gives the reward for selecting action `a` in state `s` at epoch `t`.

`policy` An integer matrix in which the `(s,t)`-th entry represents the action prescribed at decision epoch `t` by the policy under consideration for state `s`. The size and entries of this matrix should be consistent with the horizon length, states and action sets stipulated by `transition_probabilities` and `rewards`.

`rho` A number between 0 and 1 representing the reward discount factor. The default value 1 corresponds to an undiscounted model.

Details

Let π denote the Markov deterministic policy represented by `policy`. For each state s, define

$$u_t^\pi(s) = \mathbb{E}_s^\pi(\sum_{i=t}^{H-1} \rho^{i-t} R_i(X_i, d_i(X_i)) + \rho^{H-t} R_H(X_H))$$

to be the expected reward that accrues from using π between a decision epoch t and terminal epoch H (`horizon`), with $d_i(X_i)$ denoting the action prescribed at time i by π for (random) state X_i, and $R_i(X_i, d_i(X_i))$ denoting the attendant reward vector. We assume no action is taken at termination, and so the reward $R_H(X_H)$ will merely be a function of the terminal state X_H.

This function finds u_1^π, π's value function, through the recursion

$$u_t^\pi(s) = R_t(s, d_t(s)) + \rho \sum_j p_t(j|s, d_t(s)) u_{t+1}^\pi(j),$$

where the index j runs over the states, $p_t(j|s, d_t(s))$ denotes the probability of transition from s to j under action $d_t(s)$, and where we set $u_H^\pi(s) = R_H(s)$ for each state s.

Value

A numeric matrix whose `s`-th column gives the value achieved by `policy` from initial state `s`.

References

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

See Also

are_valid_finite_horizon_transition_probabilities(), are_valid_finite_horizon_rewards()

Examples

```
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2, 3))
horizon <- 5
no_objectives <- 2
MMDP <- generate_rand_MMDP(no_states, action_sets, horizon, no_objectives)
transition_probabilities <- MMDP$P
rewards <- MMDP$R
rho <- .97

# Consider this policy: for state 1, always take action 1;
# for state 2, take action 1 at odd decision epochs and action 2 at even decision epochs.
policy <- matrix(nrow = no_states, ncol = (horizon - 1), data = c(1, 1, 1, 1,
                                                                  1, 3, 1, 3), byrow = TRUE)
evaluate_finite_horizon_MMDP_markov_policy(transition_probabilities, rewards, policy, rho)
```

---

generate_rand_MMDP *Generate a random instance of a multi-objective Markov decision process*

Description

generate_rand_MMDP() returns a randomly generated finite- or infinite-horizon multi-objective Markov decision process having the requested states and action sets.

Usage

```
generate_rand_MMDP(no_states, action_sets, horizon = Inf, no_objectives)
```

Arguments

no_states	An integer greater than or equal to 2. This is the number of states in the model to be generated.
action_sets	A list of no_states integer vectors specifying each state's action set. Each vector must range from 1 to the number of actions permissible in the corresponding state.
horizon	• Inf (default): Leave at this value if the model to be generated is a stationary infinite-horizon one. • Otherwise, this is the number of decision epochs plus the terminal epoch. Has to be greater than or equal to 2.
no_objectives	An integer greater than or equal to 1. This is the number of reward functions (objectives) in the model to be generated.

Value

A list summarizing the components of the multi-objective Markov decision process that was generated in the body of the function. These components are:

- The state set `S`, an integer vector, as determined by `no_states`,
- The action sets `action_sets`, an integer list,
- The decision-making horizon `horizon`,
- A randomly generated transition probability list `P` of length one if the horizon is infinite and of length `horizon - 1` otherwise. There are three levels in `P`. In the finite-horizon case, a typical third-level entry, `P[[t]][[s]][[a]]`, is a numeric vector that gives the probability distribution of the states at epoch `t+1` if the state at epoch `t` was `s` and the action selected was `a`. For an infinite-horizon model, `P[[1]][[s]][[a]]` is to be interpreted as the stationary (i.e., time-homogeneous) probability distribution of future states if the current state is `s` and the action taken was `a`.
- A randomly generated reward list `R` of length one if the horizon is infinite and of length `horizon` otherwise. There are likewise three levels in `R`. In the finite-horizon case, a typical third-level entry, `R[[t]][[s]][[a]]`, is a numeric `no_objectives`-dimensional vector that gives the reward for selecting action `a` in state `s` at epoch `t`. A special case is the entry `R[[horizon]][[s]]`, which is a `no_objectives`-dimensional vector that gives the reward for the terminating in state `s`. In the infinite-horizon case, `R[[1]][[s]][[a]]` is the stationary reward for selecting action `a` in state `s`. There are no terminal rewards in infinite-horizon problems.

Examples

```
# Generate a two-state finite-horizon bi-objective Markov decision process
no_states <- 2
# Two actions are available in state 1 whereas three are available in state 2
action_sets <- list(c(1, 2), c(1, 2, 3))
# Let there be four decision epochs and two objectives
horizon <- 5
no_objectives <- 2
MMDP <- generate_rand_MMDP(no_states, action_sets, horizon, no_objectives)
# Inspect the randomly generated transition probabilities and rewards
MMDP$P
MMDP$R
```

---

is_valid_finite_horizon_policy

Determine whether an integer matrix represents a policy for a given class of finite-horizon problems

---

Description

`is_valid_finite_horizon_policy()` returns `TRUE` if `policy` represents a valid policy for finite-horizon problems with reward list `rewards`. In the opposite case, it returns `FALSE` together with an explanation.

Usage

```
is_valid_finite_horizon_policy(policy, rewards)
```

Arguments

policy	An integer matrix.
rewards	A numeric list providing a valid finite-horizon reward structure. See the documentation of evaluate_finite_horizon_MMDP_markov_policy() for the form required of such a list.

Details

This function juxtaposes policy against the template documented under evaluate_finite_horizon_MMDP_markov_policy and in the vignette for describing finite-horizon policies. In particular, it checks whether:

- the number of rows equals the number of states stipulated by rewards;
- the number of columns equals the length of the decision-making horizon, minus the terminal epoch, stipulated by rewards;
- the entries conform to the action sets stipulated by rewards.

Value

A boolean: TRUE if policy defines a valid Markov deterministic policy with respect to problems in which the rewards are given by rewards, FALSE otherwise.

See Also

are_valid_finite_horizon_rewards()

Examples

```
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2))
horizon <- 5
no_objectives <- 2
MMDP <- generate_rand_MMDP(no_states, action_sets, horizon, no_objectives)
rewards <- MMDP$R
# An example of a valid policy for a two-state, two-action bi-objective MDP
# with four decision epochs
policy <- matrix(nrow = no_states, ncol = (horizon - 1), data = c(2, 1, 1, 1,
                                                                  1, 2, 1, 2), byrow = TRUE)
is_valid_finite_horizon_policy(policy, rewards)
# The following policy is invalid because it prescribes no actions for epoch four
policy <- matrix(2, ncol = 3, data = c(2, 1, 1,
                                       1, 2, 1), byrow = TRUE)
is_valid_finite_horizon_policy(policy, rewards)
```

`is_valid_infinite_horizon_policy` *Determine whether an integer vector represents a stationary policy for a given class of infinite-horizon problems*

Description

`is_valid_infinite_horizon_policy()` returns `TRUE` if `stationary_policy` represents a valid stationary deterministic policy for infinite-horizon problems with reward list `stationary_rewards`. In the opposite case, it returns `FALSE` together with an explanation.

Usage

```
is_valid_infinite_horizon_policy(stationary_policy, stationary_rewards)
```

Arguments

`stationary_policy`
: An integer vector.

`stationary_rewards`
: A numeric list providing a valid infinite-horizon reward structure. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.

Details

This function juxtaposes `stationary_policy` against the template documented under `evaluate_discounted_MMDP_pure_p` for describing stationary policies. In particular, it checks whether:

- the number of entries equals the number of states implied by `stationary_rewards`;
- the entries themselves conform to the action sets implied by `stationary_rewards`.

Value

A boolean: `TRUE` if `stationary_policy` defines a valid stationary policy with respect to problems where the rewards are given by `rewards`, `FALSE` otherwise.

See Also

`are_valid_infinite_horizon_rewards()`

Examples

```
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2))
no_objectives <- 2
stationary_MMDP <- generate_rand_MMDP(no_states, action_sets, horizon = Inf, no_objectives)
stationary_rewards <- stationary_MMDP$R
```

```
# An example of a valid stationary policy: send state 1 to action 1, state 2 to action 1
stationary_policy <- c(1, 1)
is_valid_infinite_horizon_policy(stationary_policy, stationary_rewards)
# An example of an invalid stationary policy: send state 1 to action 1,
# but send state 2 to an action that falls outside its action set
stationary_policy <- c(1, 3)
is_valid_infinite_horizon_policy(stationary_policy, stationary_rewards)
```

`solve_discounted_MDP_policy_iteration` *Optimize a discounted infinite-horizon Markov decision process through policy iteration*

Description

`solve_discounted_MDP_policy_iteration()` is an adjunct to `solve_discounted_MMDP_weighting_factor()` that implements the policy iteration method for a discounted infinite-horizon MDP. It returns an optimal stationary policy together with its value function.

Usage

```
solve_discounted_MDP_policy_iteration(
  stationary_transition_probabilities,
  stationary_scalar_rewards,
  rho
)
```

Arguments

`stationary_transition_probabilities`	A numeric list describing the (stationary) transition probabilities of the infinite-horizon model under study. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.
`stationary_scalar_rewards`	A numeric list describing the model's rewards. See the documentation of `evaluate_discounted_MMDP_p` for the precise form of a finite-horizon reward list.
`rho`	A number greater than or equal to 0 but strictly less than 1 to use as a reward discount factor.

Details

Policy iteration starts from an arbitrary stationary policy then proceeds to improve on it until the policy iterates generated over the course of the algorithm stabilize. The algorithm is as follows, where the phrase "decision rule" refers to a mapping from states to actions:

- **Step 1**: Set $n = 0$, and select an arbitrary decision rule d_0.

- **Step 2**: (Policy evaluation) Calculate the value function v^n of the stationary policy that always uses d_n.
- **Step 3:** (Policy improvement) For each state s, choose $d_{n+1}(s)$ from the actions that maximize the quantity

$$r(s,a) + \rho \sum_{j \in S} p(j|s,a) v^n(j),$$

  where $r(s,a)$ is the reward for taking action a in state s and $p(j|s,a)$ is the probability of transition from s to j under a.
- **Step 4:** If $d_{n+1} = d_n$, stop: the policy that repeats d_n is optimal. Otherwise increment n by 1 and return to step 2.

When the state and action sets are finite, this procedure terminates in an optimal policy after a finite number of iterations. To implement step 2, the function calls `evaluate_discounted_MMDP_pure_policy()`. To evaluate the quantities in step 3 prior to maximization, the function calls `discounted_bellman_operator()`.

Value

A list with one integer vector and one numeric vector:

- `optimal_policy`, of length equal to the number of states as determined by `stationary_transition_probabilities` and `stationary_scalar_rewards`, defines an optimal stationary policy such that s-th entry provides the action prescribed for state s.
- `optimal_value_function`, of the same length as `optimal_policy`, gives the expected discounted reward achieved by `optimal_policy` from each initial state.

References

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

See Also

To ensure that the transition probability and reward lists passed are acceptable, use `are_valid_infinite_horizon_transit` and `are_valid_infinite_horizon_rewards()`.

Examples

```
set.seed(1234)
stationary_MDP <- generate_rand_MMDP(2, list(c(1, 2), c(1, 2, 3)), horizon = Inf, no_objectives = 1)
stationary_transition_probabilities <- stationary_MDP$P
stationary_scalar_rewards <- stationary_MDP$R
rho <- .7
solve_discounted_MDP_policy_iteration(stationary_transition_probabilities,
                                      stationary_scalar_rewards, rho)
```

`solve_discounted_MMDP_linear_programming` *Optimize a discounted infinite-horizon multi-objective Markov decision process through linear programming*

Description

`solve_discounted_MMDP_linear_programming()` implements a linear programming approach to the solution of a discounted infinite-horizon multi-objective Markov decision process. The function "maximizes" the expected discounted total reward under a fixed `initial_state_distribution` and returns all efficient stationary policies together with their value vectors.

Usage

```
solve_discounted_MMDP_linear_programming(
  stationary_transition_probabilities,
  stationary_rewards,
  rho,
  initial_state_distribution,
  max_iter = 100
)
```

Arguments

`stationary_transition_probabilities`
: A numeric list defining the (stationary) transition probabilities of the infinite-horizon model under study. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.

`stationary_rewards`
: A numeric list defining the (stationary) reward structure. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.

`rho`
: A number greater than or equal to 0 but strictly less than 1 to use as a reward discount factor.

`initial_state_distribution`
: A numeric vector, whose length equals the number of states and whose entries sum to 1.0, specifying the initial probability of each state. The linear programming approach requires that all probabilities be positive.

`max_iter`
: An integer greater than or equal to 1 giving the maximum number of iterations in the optimization phase. The default value is `100`.

Details

This function extends to the multi-objective case the linear programming approach described in Puterman (1994). The basis for this approach is that the expected discounted reward of a policy given an initial-state distribution (`initial_state_distribution`) is linear in the *state-action*

frequencies induced by that policy. A state-action frequency is the discounted total joint probability that a particular state will be visited and a particular action will be taken at least once in the process's lifetime. The collection of these frequencies forms a convex polyhedral set that is in a one-to-one correspondence with Markov randomized policies, and whose extreme points uniquely determine the model's stationary deterministic policies. Using Mifrani and Noll's (2025) adaptation of a multi-objective simplex-type algorithm by Ecker and Kouada (1978), this function enumerates those extreme points that are efficient solutions to the multi-objective linear program then calculates the corresponding (efficient) stationary policies.

Value

A list with one integer matrix and one numeric matrix:

- `efficient_pure_policies` tabulates the efficient stationary policies found through linear programming. Each row represents a policy and thus possesses as many entries as there are states.
- `efficient_value_vectors` records in its columns the value vectors generated by the policies in `efficient_pure_policies`. The length of each column is the number of objectives as determined by `stationary_rewards`.

References

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Mifrani, A., & Noll, D. (2025). Linear programming for finite-horizon vector-valued Markov decision processes. *arXiv preprint arXiv:2502.13697*.

See Also

`solve_MOLP()` for an implementation of Mifrani and Noll's version of the Ecker-Kouada multi-objective linear programming method. To ensure that the transition probability and reward lists passed are acceptable, use `are_valid_infinite_horizon_transition_probabilities()` and `are_valid_infinite_horizon_rewards()`.

Examples

```
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2, 3))
no_objectives <- 2
stationary_MMDP <- generate_rand_MMDP(no_states, action_sets, horizon = Inf, no_objectives)
stationary_transition_probabilities <- stationary_MMDP$P
stationary_rewards <- stationary_MMDP$R
rho <- .7

# Use multi-objective linear programming to solve the model
# under a uniform initial-state distribution
initial_state_distribution <- c(.5, .5)
solve_discounted_MMDP_linear_programming(stationary_transition_probabilities,
                                         stationary_rewards, rho, initial_state_distribution)
```

`solve_discounted_MMDP_policy_iteration` *Optimize a discounted infinite-horizon multi-objective Markov decision process through policy iteration*

Description

`solve_discounted_MMDP_policy_iteration()` implements a vector-valued policy iteration procedure to solve a discounted infinite-horizon multi-objective Markov decision process.

Usage

```
solve_discounted_MMDP_policy_iteration(
  stationary_transition_probabilities,
  stationary_rewards,
  rho
)
```

Arguments

`stationary_transition_probabilities`	A numeric list describing the (stationary) transition probabilities of the infinite-horizon model under study. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.
`stationary_rewards`	A numeric list describing the (stationary) reward structure. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.
`rho`	A number greater than or equal to 0 but strictly less than 1 to use as a reward discount factor.

Details

This function implements Kazuyoshi Wakuta's multi-objective extension of the standard policy iteration method for infinite-horizon MDPs. Wakuta's algorithm produces stationary deterministic (also known as pure) policies which achieve a Pareto efficient expected discounted total reward vector from any initial state.

Value

A list with two components:

- An integer matrix with one column per state in which the rows define the efficient stationary policies found by policy iteration. The `s`-th entry in each row gives the action prescribed by the row's policy for state `s`.
- A list containing one numeric matrix per efficient policy that records the policy' value function. The column indices of each matrix represent the initial states and the columns themselves give the value vectors achieved from those states.

References

Wakuta, K. (1995). Vector-valued Markov decision processes and the systems of linear inequalities. *Stochastic processes and their applications, 56*(1), 159-169.

See Also

`solve_discounted_MDP_policy_iteration()` for an implementation of the standard policy iteration method in a discounted single-objective MDP. To ensure that the transition probability and reward lists provided are acceptable, use `are_valid_infinite_horizon_transition_probabilities()` and `are_valid_infinite_horizon_rewards()`.

Examples

```
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2, 3))
no_objectives <- 2
stationary_MMDP <- generate_rand_MMDP(no_states, action_sets, horizon = Inf, no_objectives)
stationary_transition_probabilities <- stationary_MMDP$P
stationary_rewards <- stationary_MMDP$R
rho <- .7

# Let's locate the efficient stationary policies for this infinite-horizon model
solution <- solve_discounted_MMDP_policy_iteration(stationary_transition_probabilities,
                                                   stationary_rewards, rho)
solution$policies
# Inspect their value functions
solution$value_functions
```

---

solve_discounted_MMDP_weighting_factor

Optimize a discounted infinite-horizon multi-objective Markov decision process through the weighting factor approach

---

Description

`solve_discounted_MMDP_weighting_factor()` generates efficient stationary policies for a discounted infinite-horizon multi-objective Markov decision process using the weighting factor approach.

Usage

```
solve_discounted_MMDP_weighting_factor(
  stationary_transition_probabilities,
  stationary_rewards,
  rho,
  no_iterations = 100
)
```

Arguments

`stationary_transition_probabilities`
: A numeric list describing the (stationary) transition probabilities of the infinite-horizon model under study. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.

`stationary_rewards`
: A numeric list describing the (stationary) reward structure. See the documentation of `evaluate_discounted_MMDP_pure_policy()` for the form required of such a list.

`rho`
: A number greater than or equal to 0 but strictly less than 1 to use as a reward discount factor.

`no_iterations`
: The number of weighting factors, each factor being a numeric vector with one entry per objective, to be used. The default value is `100`.

Details

In a discounted infinite-horizon MMDP with finite state and action spaces, there exist stationary policies which are efficient. Furthermore, each such policy maximizes some positive linear combination of the objectives (Wakuta, 1982). Conversely, if we consider the problem of maximizing a positive linear combination (any one would do) of the objectives, we get a single-objective MDP for which any (uniformly) optimal policy is also (uniformly) efficient in the original multi-objective model. This function follows this approach, sampling `no_iterations` positive weight vectors at random then solving the resulting MDPs with the standard policy iteration method.

Value

A list with one integer matrix and one numeric sublist:

- `efficient_policies`, one row per efficient policy detected and one column per state, tabulates the policies found. The (p, s)-th entry specifies the action prescribed by policy p for state s.
- `efficient_value_functions`, comprised of one numeric matrix per policy in `efficient_policies`, records each policy's value function. The s-th column of a value function matrix contains the expected discounted reward attained from initial state s.

References

Wakuta, K. (1995). Vector-valued Markov decision processes and the systems of linear inequalities. *Stochastic processes and their applications, 56*(1), 159-169.

See Also

`solve_discounted_MDP_policy_iteration()`, `evaluate_discounted_MMDP_pure_policy()`. To ensure that the transition probability and reward lists passed are acceptable, use `are_valid_infinite_horizon_transitior` and `are_valid_infinite_horizon_rewards()`.

Examples

```
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2, 3))
no_objectives <- 2
stationary_MMDP <- generate_rand_MMDP(no_states, action_sets, horizon = Inf, no_objectives)
stationary_transition_probabilities <- stationary_MMDP$P
stationary_rewards <- stationary_MMDP$R
rho <- .7
solve_discounted_MMDP_weighting_factor(stationary_transition_probabilities, stationary_rewards, rho)
```

---

solve_finite_horizon_MDP_backward_induction

Solve a standard finite-horizon Markov decision process through dynamic programming

---

Description

`solve_finite_horizon_MDP_backward_induction()` implements the standard backward induction algorithm in order to determine the optimal value function and an optimal policy for a single-objective finite-horizon Markov decision process.

Usage

```
solve_finite_horizon_MDP_backward_induction(
  transition_probabilities,
  scalar_rewards,
  rho = 1
)
```

Arguments

`transition_probabilities`	A numeric list defining the transition probabilities of the model to be solved. See the documentation of `evaluate_finite_horizon_MMDP_markov_policy()` for the structure expected of a finite-horizon transition probability list.
`scalar_rewards`	A numeric list defining the rewards. See the documentation of `evaluate_finite_horizon_MMDP_markov` for the structure expected of a finite-horizon reward list.
`rho`	A number between (and including) 0 and 1 to use as the reward discount factor. The default value is 1 (no discount).

Value

A list of one numeric matrix and one integer matrix:

- `optimal_values` has one row per state and one column per epoch. The `(s, t)`-th entry records the largest expected total reward that can be achieved between time `t` and process termination if the initial state is `s`.
- `optimal_policy` represents an optimal policy in the manner of `evaluate_finite_horizon_MMDP_markov_policy()`.

On backward induction

In keeping with the notation of evaluate_finite_horizon_MMDP_markov_policy(), let $u_t^*(s)$ denote the maximum expected (possibly discounted) reward accrued from time t onward if the state at that time is s. It is shown in Puterman (2014) and elsewhere that the optimal value function u_1^* can be determined from the inductive relationship

$$u_t^*(s) = \max_a (r_t(s,a) + \rho \sum_j p_t(j|s,a) u_{t+1}^*(j)),$$

where the index a runs over the action set of state s, and where $u_H^*(s) = r_H(s)$. In a finite-horizon MDP with finite state and action sets, there always exists an optimal policy which is Markov deterministic. In fact, if for each decision epoch t and each state s we record those actions a that cause the above maximum to be attained, then we have constructed all optimal Markov deterministic policies. The present function calls MDP_update() to carry out the maximization for each epoch and constructs an optimal policy in the iterative manner described.

On rewards

Inasmuch as this function is intended for single-objective problems, a reward scalar_rewards[[t]][[s]][[a]] (when t is a decision epoch) or scalar_rewards[[horizon]][[s]] (horizon being the length of scalar_rewards) should be a number.

On the optimal policy

Standard Markov decision theory assures us that backward induction in finite-horizon MDPs having finite states and actions constructs policies that are optimal at every stage of decision making. In particular, optimal_policy achieves the largest expected reward not only between time 1 and termination, but between any time t and termination.

See Also

To ensure that the transition probability and reward lists provided are acceptable, use are_valid_finite_horizon_transit
and are_valid_finite_horizon_rewards().

Examples

```
set.seed(1234)
MDP <- generate_rand_MMDP(2, list(c(1, 2), c(1, 2, 3)), 5, no_objectives = 1)
transition_probabilities <- MDP$P
scalar_rewards <- MDP$R
rho <- .97
solve_finite_horizon_MDP_backward_induction(transition_probabilities, scalar_rewards, rho)
```

`solve_finite_horizon_MMDP_backward_induction` *Optimize a finite-horizon multi-objective Markov decision process through vector-valued dynamic programming*

Description

`solve_finite_horizon_MMDP_backward_induction()` finds the Pareto efficient values of a finite-horizon multi-objective Markov decision process by solving a vector-valued extension of the dynamic programming equations.

Usage

```
solve_finite_horizon_MMDP_backward_induction(
  transition_probabilities,
  rewards,
  rho = 1,
  method = "heuristic"
)
```

Arguments

`transition_probabilities`	A numeric list specifying the finite-horizon transition probabilities of the model under study. See the documentation of `evaluate_finite_horizon_MMDP_markov_policy()` for a description of what such a list should look like.
`rewards`	A numeric list defining the model's reward structure. See the documentation of `evaluate_finite_horizon_MMDP_markov_policy()` for a description of what such a list should look like.
`rho`	A number between (and including) 0 and 1 to use as the reward discount factor. The default value is 1 (no discount).
`method`	A string indicating the method by which the efficiency equations are solved: `"exact"` for enumeration or `"heuristic"` for approximate solution using a genetic algorithm (default).

Details

This function solves a multi-objective counterpart to the optimality equations involved in `solve_finite_horizon_MDP_back`
In order that we define the new equations and the manner in which they characterize the efficient values of a finite-horizon MMDP, it is necessary to expand the policy space to allow for a special kind of policy which is not Markov. If $h_t = (s_1, \ldots, s_t)$ denotes a sequence of states observed prior to and including time t, then we shall be interested in policies that map each such sequence to some action permissible in s_t. Notice that this policy space subsumes Markov deterministic policies, which depend on state histories only through the current state. For any policy π, let $u_t^{\pi}(s)$ denote the expected total reward vector it achieves from time t onward if the state at that time is s. The

set of all such expected reward vectors is denoted by $U_t(s)$. Then, if $U_t^*(s)$ represents the efficient subset of $U_t(s)$, Mifrani (2024) shows that $U_t^*(s)$ can be determined from the recursion

$$U_t^*(s) = e\left(\bigcup_a \left(\{R_t(s,a)\} \bigoplus \sum_j p_t(j|s,a) U_{t+1}^*(j)\right)\right),$$

where $e(X)$ denotes the efficient subset of some vector set X, and $\bigoplus$ denotes a sum set.

This function finds $U_1^*(s)$ for each initial state s by carrying out the above recursion backward in time, starting from the boundary condition that the efficient value sets at termination, $U_H^*(s)$, reduce to the terminal rewards $R_H(s)$.

To compute the efficient subsets involved at each epoch, the function calls either `exact_MMDP_update()` or `heuristic_MMDP_update()`, depending on whether the `method` sought is enumerative or heuristic. The former relies on `efficient_subset_sort_prune()`, whereas the latter uses `nsga2R::nsga2R()`, an implementation of the popular Non-dominated Sorting Genetic Algorithm II (more commonly known as NSGA-II).

Value

A list with three components:

- `efficient_values`, a numeric list the length of `rewards[[t]]` / `transition_probabilities[[t]]` (i.e., the cardinality of the state set) in which each component is a numeric matrix representing the efficient value set $U_1^*(s)$ relative to some starting state s (see the Details section). The column vectors of each matrix give the efficient values that can be achieved from the corresponding state, or an approximation of these values in case the solution procedure is `heuristic`. The length of these column vectors is the number of objectives.
- `compromise_solution`, a numeric list the length of `efficient_values` that stores, for each initial state, the "compromise" value obtained by application of `compromise_solution()` to the columns of the matrix in `efficient_values` corresponding to that state. See the documentation of `compromise_solution()` for more information.
- `time`, the CPU time in seconds.

References

Mifrani, A. (2025). A counterexample and a corrective to the vector extension of the Bellman equations of a Markov decision process. *Annals of Operations Research, 345*(1), 351-369.

See Also

To ensure that the transition probability and reward lists provided are acceptable, use `are_valid_finite_horizon_transit`
and `are_valid_finite_horizon_rewards()`.

Examples

```
set.seed(1234)
MMDP <- generate_rand_MMDP(no_states = 2, action_sets = list(c(1, 2), c(1, 2, 3)),
                           horizon = 5, no_objectives = 2)
transition_probabilities <- MMDP$P
rewards <- MMDP$R
solve_finite_horizon_MMDP_backward_induction(transition_probabilities, rewards)
```

`solve_finite_horizon_MMDP_linear_programming` *Optimize a finite-horizon multi-objective Markov decision process through linear programming*

Description

`solve_finite_horizon_MMDP_linear_programming()` implements a linear programming approach to the solution of a finite-horizon multi-objective Markov decision process. The function "maximizes" the expected total reward vector under a fixed distribution of initial states `initial_state_distribution`. It returns all efficient Markov deterministic policies along with the values they generate.

Usage

```
solve_finite_horizon_MMDP_linear_programming(
  transition_probabilities,
  rewards,
  initial_state_distribution,
  max_iter = 100
)
```

Arguments

`transition_probabilities`
: A numeric list specifying the transition probabilities of the finite-horizon model under study. See the documentation of `evaluate_finite_horizon_MMDP_markov_policy()` for the form required of such a list.

`rewards`
: A numeric list specifying the reward structure. See the documentation of `evaluate_finite_horizon_MM` for the form required of such a list.

`initial_state_distribution`
: A numeric vector, whose length equals the number of states and whose entries sum to 1.0, specifying the initial probability of each state. The linear programming approach requires that all probabilities be positive.

`max_iter`
: An integer greater than or equal to 1 giving the maximum number of iterations in the optimization phase. The default value is `100`.

Details

This function implements the linear programming technique proposed by Mifrani and Noll (2025). The technique rests on the observation that the finite-horizon expected total reward of a policy under an initial-state distribution (`initial_state_distribution`) is linear in the *state-action frequencies* induced by that policy. A state-action frequency $x_{\pi,t}(s,a)$ is the joint probability that the process will enter state s at time t and select action a under policy π. The collection of these frequencies forms a convex polyhedral set that is in a one-to-one correspondence with Markov randomized policies, and whose extreme points uniquely determine the model's Markov deterministic policies. Using Mifrani and Noll's adaptation of a multi-objective simplex-type algorithm by Ecker

and Kouada (1978), this function enumerates those extreme points that are efficient solutions to the multi-objective linear program then calculates the corresponding (efficient) policies.

Value

A list of one integer sublist and one numeric matrix:

- `efficient_policies` contains integer matrices representing the efficient policies detected by linear programming.
- `efficient_value_vectors` records in its columns the value vectors generated under `initial_state_distribution` by the policies in `efficient_policies`. The length of each column is the number of objectives as determined by `rewards`.

References

Mifrani, A., & Noll, D. (2025). Linear programming for finite-horizon vector-valued Markov decision processes. *arXiv preprint arXiv:2502.13697*.

See Also

`solve_MOLP()` for an implementation of Mifrani and Noll's version of the Ecker-Kouada multi-objective linear programming technique. To ensure that the transition probability and reward lists passed are acceptable, use `are_valid_finite_horizon_transition_probabilities()` and `are_valid_finite_horizor`

Examples

```
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2, 3))
no_objectives <- 2
horizon <- 5
MMDP <- generate_rand_MMDP(no_states, action_sets, horizon, no_objectives)
transition_probabilities <- MMDP$P
rewards <- MMDP$R
# Use multi-objective linear programming to solve the model
# under a uniform initial-state distribution
initial_state_distribution <- c(.5, .5)
solve_finite_horizon_MMDP_linear_programming(transition_probabilities,
                                        rewards, initial_state_distribution)
```

---

solve_finite_horizon_MMDP_weighting_factor

Optimize a finite-horizon multi-objective Markov decision process through the weighting factor approach

---

Description

`solve_finite_horizon_MMDP_weighting_factor()` finds Pareto efficient Markov deterministic policies for a finite-horizon multi-objective Markov decision process, with or without discount, using the weighting factor approach.

Usage

```
solve_finite_horizon_MMDP_weighting_factor(
  transition_probabilities,
  rewards,
  rho = 1,
  no_iterations = 100
)
```

Arguments

`transition_probabilities`	A numeric list defining the transition probabilities of the finite-horizon model under study. See the documentation of `evaluate_finite_horizon_MMDP_markov_policy()` for the detailed structure of such a list.
`rewards`	A numeric list defining the model's reward structure. See the documentation of `evaluate_finite_horizon_MMDP_markov_policy()` for the detailed structure of such a list.
`rho`	A number between 0 and 1 representing the reward discount factor. The default value 1 yields an undiscounted model.
`no_iterations`	The number of weighting factors, each factor being a numeric vector with one positive entry per objective, to be used. One single-objective MDP will be solved per factor. The default value is `100`.

Details

The premise of the weighting factor approach for finite-horizon problems is identical to the infinite-horizon case as documented under `solve_discounted_MMDP_weighting_factor()`, except that the efficient policies that are generated need not be stationary.

Value

A list with two sublists:

- `efficient_policies` is a list where each efficient policy detected by the weighting factor approach occupies an integer matrix. For each of these matrices, the (s, t)-th entry specifies the action prescribed for state s at time t.
- `efficient_value_functions`, comprised of one numeric matrix per policy in `efficient_policies`, records each policy's value function. The s-th column of a value function matrix contains the expected discounted reward attained from initial state s.

See Also

`solve_discounted_MMDP_weighting_factor()` for an analogous implementation in the infinite-horizon case. To ensure that the transition probability and reward lists provided are acceptable, use `are_valid_finite_horizon_transition_probabilities()` and `are_valid_finite_horizon_rewards()`.

Examples

```
set.seed(1234)
no_states <- 2
action_sets <- list(c(1, 2), c(1, 2, 3))
no_objectives <- 2
horizon <- 5
MMDP <- generate_rand_MMDP(no_states, action_sets, horizon, no_objectives)
transition_probabilities <- MMDP$P
rewards <- MMDP$R
rho <- .97
solve_finite_horizon_MMDP_weighting_factor(transition_probabilities, rewards, rho)
```

---

solve_MOLP *Solve a multi-objective linear programming problem by a simplex-type method*

Description

`solve_MOLP()` is an adjunct to `solve_discounted_MMDP_linear_programming()` that enumerates the efficient basic feasible solutions of a multi-objective linear programming problem subject to equality and nonnegativity constraints.

Usage

```
solve_MOLP(C, A, b, max_iter)
```

Arguments

`C`	A numeric matrix whose i-th row contains the coefficient vector for linear objective function i.
`A`	A numeric matrix, one row per constraint and one column per variable, giving the constraint coefficients.
`b`	A numeric vector giving the right-hand side coefficients.
`max_iter`	An integer greater than or equal to 1 that places an upper limit on the number of iterations performed by the linear programming procedure.

Details

This function implements a particular version of Ecker and Kouada's (1978) procedure for enumerating the efficient basic feasible solutions of a multi-objective linear (maximization) program where candidate points are tested for efficiency using a proposition from Evans and Steuer (1973). The procedure requires an initial efficient extreme point, which can be found by maximizing a positive linear combination of the objectives.

Value

A list with one integer matrix and one numeric matrix:

- `efficient_bfs`, whose rows contain the efficient basic feasible solutions—which are vectors of length equal to the number of variables—found;
- and `values`, whose columns contain the objective value vectors generated by the solutions in `efficient_bfs`.

References

Ecker, J. G., & Kouada, I. A. (1978). Finding all efficient extreme points for multiple objective linear programs. *Mathematical programming, 14*(1), 249-261.

Evans, J. P., & Steuer, R. E. (1973). A revised simplex method for linear multiple objective programs. *Mathematical programming, 5*(1), 54-72.

Mifrani, A., & Noll, D. (2025). Linear programming for finite-horizon vector-valued Markov decision processes. *arXiv preprint arXiv:2502.13697*.

See Also

`solve_discounted_MMDP_linear_programming()`

Examples

```
# Consider the multi-objective linear programming problem
#           Maximize 4*x_1 - x_2 + 3*x_3
#                    -2*x_1 + 5*x_2 - 0.5*x_3
#           subject to 2*x_1 + 3*x_2 - x_3 = 12
#                      x_1 - x_2 + 3*x_3 = 3
#                      x_1, x_2, x_3 nonnegative



# Criteria matrix
C <- matrix(c(4, -1, 3,
              -2, 5, -0.5), nrow = 2, ncol = 3, byrow = TRUE)

# Constraint matrix (excluding nonnegativity)
A <- matrix(c(2, 3, -1,
             1, -1, 3), nrow = 2, ncol = 3, byrow = TRUE)
# Right-hand side vector
b <- c(12, 3)
# Solve the multi-objective linear programming problem,
# display the efficient basic feasible solutions
result <- solve_MOLP(C, A, b, max_iter = 50)
result$efficient_bfs
result$values
```

sum_set *Calculate the sum set (Minkowski sum) of two or more sets of vectors*

Description

sum_set() is an adjunct to exact_MMDP_update() and heuristic_MMDP_update() that forms the sum set of two or more sets viewed as matrices whose columns represent the sets' elements.

Usage

```
sum_set(X)
```

Arguments

X	A list of numeric matrices representing Euclidean sets. Each column in a matrix represents an element of the set of which the matrix is a representation. The matrices must have an identical number of rows, which is to say that the underlying sets must have the same dimension.

Value

A numeric matrix representing the sum set of the sets in X.

Examples

```
set_1 <- matrix(c(1, 2, 0, 0, 3, -6), nrow = 2, ncol = 3)
set_2 <- matrix(c(5, -2, 7, 0), nrow = 2, ncol = 2)
X <- list(set_1, set_2)
sum_set(X) # Returns the sum set of set_1 and set_2
```

value_function_domination_sets *Compare two or more vector-valued value functions*

Description

value_function_domination_sets() is an adjunct to solve_discounted_MMDP_policy_iteration() that compares a vector-valued function V with a set of such functions to determine which of them dominate V, are dominated by V or are equal to V.

Usage

```
value_function_domination_sets(V, value_functions)
```

Arguments

V	A numeric matrix in which the s-th column gives the image of s under the vector-valued function encoded in V.
value_functions	A list of numeric matrices, having the same size as V, with which V is to be compared.

Details

Dominance here is taken with respect to the product order induced by the "larger-is-better" componentwise order over column vectors. That is, if U denotes some matrix in value_functions, then U dominates V if U[, s] is componentwise greater than or equal to V[, s] for all columns s and there exists a column j such that U[, j] dominates V[, j] componentwise.

Value

A list comprising, in this order,

- the matrices in value_functions that dominate V,
- those that are dominated by V,
- and those that are identical to V.

Examples

```
V <- matrix(c(5, 5, 3, 1), nrow = 2)
U1 <- matrix(c(4, 4, 2, 0), nrow = 2) # Dominated by V
U2 <- matrix(c(6, 5, 4, 2), nrow = 2) # Dominates V
U3 <- V # Identical to V
value_functions <- list(U1, U2, U3)
value_function_domination_sets(V, value_functions)
```

Index