Package 'mstATA'

May 9, 2026

Title Automated Test Assembly for Multistage Tests Using Mixed-Integer Linear Programming

Version 0.1.0

Description Provides a suite of mixed-integer linear programming (MILP) model builders and solvers—including 'Gurobi', 'HiGHS', 'Symphony', 'GNU Linear Programming Kit (GLPK)', and 'lpSolve'—for automated test assembly (ATA) in multistage testing (MST). Offers filtering of decision variables through item–module eligibility and the application of explicit bounds to simplify the MILP model and accelerate the optimization process. Supports bottom up, top down, and hybrid assembly strategies; enemy-item and enemy-stimulus exclusions; stimulus all in/all out or partial selection; anchor item/stimulus specification; and item exposure control. Accommodates both single-objective and multi-objective optimization ('weighted sum', 'maximin', 'capped maximin', 'minimax', and 'goal programming'). Enables simultaneous assembly of multiple panels with item and stimulus content balancing and exposure control. Provides analytical evaluation of assembled MST performance within seconds. Includes tools for diagnosing infeasible optimization models by systematically identifying sources of infeasibility and reformulating models with slack variables to restore feasibility.Methods implemented in this package build on established work in optimal test assembly (van der Linden, 2005 <doi:10.1007/0-387-29054-0>), itemset constrained test assembly (van der Linden, 2000 <doi:10.1177/01466210022031697>), hybrid assembly (Xiong, 2018 <doi:10.1177/0146621618762739>), recursion-based analytic methods (Lim et al., 2021 <doi:10.1111/jedm.12276>), and classification evaluation (Rudner, 2000 <doi:10.7275/an9m-2035>; Rudner, 2005 <doi:10.7275/56a5-6b14>).

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.0)

LazyData true

Config/testthat/edition 3

Imports stats, utils, dplyr, rlang, methods, Matrix, ggplot2

Suggests testthat (>= 3.0.0),knitr,rmarkdown, kableExtra,highs,lpSolveAPI,Rglpk,Rsymphony,irtQ,mstR,numDeriv,ggformula,ggpubr

VignetteBuilder knitr

URL https://github.com/Hongchen030/mstATA

BugReports https://github.com/Hongchen030/mstATA/issues

NeedsCompilation no

Author Hong Chen [aut, cre]

Maintainer Hong Chen <hongchen030@gmail.com>

Repository CRAN

Date/Publication 2026-03-23 10:00:02 UTC

Contents

`analytic_mst_classification` *Analytic classification accuracy and consistency for MST panels*

Description

Once the analytical CSEMs are derived based on recursion-based approach (Lim, 2000), the evaluation of MST performance using classification metrics can be estimated using Rudner's method (2000,2005). The implementation follows the same logic as `irtQ::cac_rud()`.

Usage

```
analytic_mst_classification(decision_theta_cuts, eval_tb, theta_weight)
```

Arguments

`decision_theta_cuts`	Numeric vector of classification cut theta values. These define $K + 1$ ordered classification levels.
`eval_tb`	A data frame containing at least the columns `theta` and `csem`. It can be created by analytic_mst_precision().
`theta_weight`	A data frame containing the population distribution over the ability grid. It must have the following columns: **theta** Numeric ability grid. This must match `eval_tb$theta` exactly. **w** Non-negative population weights corresponding to each ability value. The `theta_weight` object can be generated using gen_weight().

Details

This function evaluates classification performance without simulation. Given a set of decision cuts and conditional standard errors across a fixed ability grid, classification accuracy and consistency are computed analytically by integrating conditional classification probabilities over a population ability distribution.

The procedure consists of the following steps:

1. **Define true classification levels:** Each ability value θ on the evaluation grid is assigned to a true classification level based on the specified cut theta values.
2. **Compute conditional classification probabilities:** For each θ, the probability of being classified into each level is computed assuming a normal distribution for the reported ability estimate:
$$\hat{\theta} \mid \theta \sim \mathcal{N}(\theta,\, CSEM(\theta)^2).$$
3. **Conditional accuracy and consistency:**
Conditional classification accuracy is defined as
$$P(\hat{C} = C \mid \theta)$$

where C is the true classification level.

Conditional consistency is defined as

$$\sum_k P(\hat{C} = k \mid \theta)^2$$

representing the probability that two independent parallel administrations yield the same classification.

4. **Marginalization over the population:**
   Conditional indices are integrated over a population ability distribution using externally supplied weights, yielding marginal classification accuracy, marginal classification consistency, and a confusion matrix.

Value

A list with the following components:

confusion A matrix giving the joint probability of true and expected classification levels. Rows correspond to true levels; columns correspond to expected levels.

marginal A data frame of marginal classification accuracy and consistency by true level, including an overall (marginal) row.

conditional A data frame of conditional accuracy and consistency at each ability value θ.

prob.level A data frame of conditional classification probabilities for each level at each θ.

cutscore The classification cut theta values used in the analysis.

Mathematical Formulation

Let θ_i denote an ability grid point and $CSEM(\theta_i)$ the corresponding conditional standard error. For classification cut scores $c_1, \ldots, c_K$, define level boundaries $(-\infty, c_1, \ldots, c_K, \infty)$.

The probability of classifying an examinee with true ability θ_i into level k is

$$P(\hat{C} = k \mid \theta_i)$$

$$= \Phi(\frac{(u_k - \theta_i)}{CSEM(\theta_i)}) - \Phi(\frac{(\ell_k - \theta_i)}{CSEM(\theta_i)})$$

where (ℓ_k, u_k) is the interval for level k.

Conditional accuracy at θ_i is

$$P(\hat{C} = C \mid \theta_i)$$

and conditional consistency is

$$\sum_{k=1}^{K+1} P(\hat{C} = k \mid \theta_i)^2$$

Let $w(\theta_i) \geq 0$ be population weights on the grid, with $\sum_{i=1}^{I} w(\theta_i) = 1$.

The marginal (population) indices are the discrete weighted averages of the conditional quantities:

$$MarginalAccuracy = \sum_{i=1}^{I} w(\theta_i) P(\hat{C} = C(\theta_i) \mid \theta_i)$$

where $C(\theta_i)$ is the true level implied by θ_i and the cut scores.

$$MarginalConsistency = \sum_{i=1}^{I} w(\theta_i) \sum_{k=1}^{K+1} P(\hat{C} = k \mid \theta_i)^2$$

If $w(\theta_i)$ approximates a density $g(\theta)$, these correspond to the integrals $\int g(\theta) \cdot d\theta$.

References

Rudner, L. M. (2000). Computing the expected proportions of misclassified examinees. *Practical Assessment, Research, and Evaluation*, 7(1). doi:10.7275/an9m-2035

Rudner, L. M. (2005). Expected classification accuracy. *Practical Assessment, Research, and Evaluation*, 10(1). doi:10.7275/56a5-6b14

`analytic_mst_precision`

Evaluate MST designs via recursive score-distribution propagation

---

Description

Computes conditional bias and conditional standard error of measurement (CSEM) for a multistage test (MST) panel using a recursion-based evaluation method. The implementation follows the logic of `irtQ::reval_mst()`, propagating score distributions stage by stage conditional on true ability θ, applying routing rules, and mapping final scores to reported ability estimates via inverse test characteristic curves (TCCs).

Usage

```
analytic_mst_precision(
  design,
  exclude_pathways = NULL,
  assembled_panel,
  item_par_cols,
  model_col,
  D = 1,
  theta = seq(-3, 3, 0.1),
  range_tcc = c(-5, 5),
  rdps,
  tol = 1e-04
)
```

Arguments

design	A character string specifying the MST design. The string encodes the number of modules per stage and may use commas, dashes, or slashes as separators. For example, "1-3-3", "1,3,3", and "1/3/3" all define an MST with 1 module in stage 1, 3 modules in stage 2, and 3 modules in stage 3.
exclude_pathways	Optional character vector specifying disallowed pathways. Each element must be a string of the form "x-y-z", where each number refers to a module index at the corresponding stage. If NULL (default), all pathways implied by the design are allowed. This option is commonly used to prohibit extreme routing transitions (e.g., from very easy to very hard modules).
assembled_panel	A "mstATA_panel" object returned by assembled_panel(). All panels share identical module and pathway structure.
item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in model_col column: "1PL", "RASCH", "2PL", "3PL","4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names that contain the required item parameters for the corresponding model.
model_col	A character string specifying the column name in either assembled_panel$ItemsInModules or assembled_panel$ItemsInPathways data frames that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: "1PL", "RASCH", "2PL", "3PL","4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM".
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.
theta	Numeric vector specifying the ability grid used to construct ICCs. Defaults are -3 to 3 with 0.1 interval.
range_tcc	Numeric vector giving the search interval for inverse TCC evaluation (length = 2). Defaults is (-5,5).
rdps	List of routing cuts on the theta scale.
tol	A convergence tolerance used for inverse TCC calculations. Default is 1e-4.

Details

1. Recursion-based analytic evaluation

The analytic evaluation proceeds in four conceptual steps:

1. **Module score distributions**
   For each module, the conditional score distribution $P(S_m \mid \theta)$ is computed from item response category probabilities.
2. **Stage-wise recursion with routing**
   Joint (cumulative) score distributions are built recursively across stages. At each stage, examinees are deterministically assigned to next-stage modules based on the comparison between their ITCC estimates and the predefined cut points for that stage.

3. **Exact joint score propagation**

   For each valid routing branch, the cumulative score distribution is updated by convolving the previous-stage score distribution with the score distribution of the next module.

4. **Inverse TCC and conditional moments**

   Final pathway-specific score distributions are mapped to reported ability estimates via inverse test characteristic curves (ITCCs). The conditional mean, conditional bias and conditional variance (CSEM) are computed at each true ability value θ.

2. Conditional bias and conditional SEM

Let S denote the cumulative test score and $\hat{\theta}(S)$ the ITCC estimator.

For each true ability value θ, this function computes:

$$\mu(\theta) = E(\hat{\theta} \mid \theta)$$
$$\sigma^2(\theta) = Var(\hat{\theta} \mid \theta)$$

where the expectation is taken with respect to the exact joint score distribution induced by the MST routing rules.

Value

A list with components:

prefix_table prefix-to-module lookup tables by stage.

eq_theta ITCC estimates for each stage.

cdist_by_mod Module-level score distributions.

joint_dist Stage-wise joint score distributions by prefix.

eval_tb A data frame with columns `theta`, `mu`, `sigma2`, `bias`, and `csem`.

Mathematical Formulation

Let $\theta \in \Theta$ denote a true ability value on a discrete grid. Consider a multistage test (MST) consisting of T stages, where at each stage exactly one module is administered according to deterministic routing rules that map observed cumulative scores to modules.

(a) Module score distributions

For module m, let S_m denote the module score. Conditional on ability θ, the module score distribution is

$$P(S_m = s \mid \theta)$$

which is obtained by analytically combining item response category probabilities under the specified IRT model. This distribution is computed independently for each module using module_score_dist().

(b) Stage-wise recursion with routing

Let $S^{(t)}$ denote the cumulative score after stage t. At each stage t, examinees are assigned to a next-stage module m_{t+1} based on the comparison between the ITCC estimates and the cut theta points.

The routing rule can be written as a deterministic mapping

$$m_{t+1} = r_t(S^{(t)})$$

where $r_t(\cdot)$ maps observed cumulative scores to modules at stage $t+1$.

(c) Exact joint score propagation

For a given routing branch induced by the realized cumulative score, the cumulative score evolves as

$$S^{t+1} = S^t + S_m^{t+1}$$

Conditional on θ, the joint score distribution is updated analytically via convolution:

$$P(S^{t+1} = s \mid \theta) = \sum_u P(S^t = u \mid \theta) P(S_m^{t+1} = s - u \mid \theta)$$

where the sum is taken over all reachable values of $S^{(t)}$ along the routing branch.

(d) Inverse TCC and conditional moments

Let $S^{(t)}$ denote the final cumulative test score, and let $\hat{\theta}(S)$ be the reported ability estimate obtained by applying the inverse test characteristic curve (TCC) associated with the administered pathway determined by the routing rules.

For each true ability value θ, the conditional mean and conditional variance of the reported ability are defined as

$$\mu(\theta) = E[\hat{\theta} \mid \theta]$$

$$\sigma^2(\theta) = Var[\hat{\theta} \mid \theta]$$

where the expectation is taken with respect to the exact joint distribution $P(S \mid \theta)$ induced by the MST routing rules.

Explicitly,

$$\mu(\theta) = \sum_s \hat{\theta}(s)\, P(S = s \mid \theta)$$

$$\sigma^2(\theta) = \sum_s (\hat{\theta}(s) - \mu(\theta))^2\, P(S = s \mid \theta)$$

References

Lim, H., Davey, T., & Wells, C. S. (2020). A recursion-based analytical approach to evaluate the performance of MST. *Journal of Educational Measurement*, 58(2), 154–178. doi:10.1111/jedm.12276

See Also

`compute_icc()`

`module_score_dist()`

`joint_module_score_dist()`

`inverse_tcc()`

`assembled_panel` *Assemble Selected Items into MST panel*

Description

Constructs a panel-centric representation of an assembled multistage panel (MST) from the solver output. The function extracts binary decision variables, maps them to items in the item pool, and organizes the results by panel, module, and pathway.

This function is typically called after `solve_model()` and is intended for post-solution interpretation, diagnostics, and reporting.

Usage

```
assembled_panel(x, result)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`result`	A solver result returned by `solve_model()`, containing components `model` and `solution`.

Details

The workflow is:

1. Extract item-module-panel binary decision variables with value 1 from the solver solution.
2. Parse decision variable names of the form `x[module_id, item_id, panel_id]`.
3. Map internal item indices (`item_id`) to item-module-panel identifiers.
4. Merge selected items with the item pool to attach item attributes.
5. For each panel:
   - Construct `ItemsInModules`, listing items selected in each module.
   - Construct `ItemsInPathways`, listing items associated with each pathway based on the pathway-module mapping.

If pathway or module labels are available in `x$PathwayIndex` and `x$ModuleIndex`, labels for modules/pathways are attached.

Value

An object of S3 class `mstATA_panel` with a named list with one element per panel: Panel_1, Panel_2, ...Each panel is a list with components:

ItemsInModules A data frame containing selected items grouped by module. Columns include `item_name`, `item_id`, `varname`, `module_id`, and all item attributes from the item pool.

ItemsInPathways A data frame containing selected items grouped by pathway. Columns include `item_name`, `item_id`, `varname`, `pathway_id`, and all item attributes from the item pool.

See Also

`solve_model()`

`binary_minimax_panel` *Precomputed MST Panel: Binary_Minimax Formulation*

Description

A precomputed multistage testing (MST) panel assembled using the minimax objective formulation (two_dev) described in the *Formulation for Multiple Objectives* vignette.

Usage

```
binary_minimax_panel
```

Format

An object of class `mstATA_panel`.

Details

This object is included to avoid long solver runtimes during vignette building and package checking.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

`BU13_panel` *Precomputed Bottom-up MST Assembly Example (BU13)*

Description

A precomputed multistage testing (MST) panel assembled using the bottom-up strategy demonstrated in the *Three ATA Strategies* Vignette.

Usage

```
BU13_panel
```

Format

An object of class `mstATA_panel`.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

`capped_maximin_obj` *Capped maximin objective*

Description

The **capped maximin** strategy balances multiple competing objectives by maximizing a common lower bound across multiple objective terms, while simultaneously limiting how far any single objective term may exceed that bound.

Compared to the classical maximin formulation–where overflow control may be imposed through a user-specified upper bound δ–the capped maximin strategy treats δ as a decision variable and optimizes it jointly with the common floor value y. The resulting objective to be maximized, $(y-\delta)$, balances improving the worst-performing objective while penalizing excessive dominance by any single objective.

Usage

```
capped_maximin_obj(x, multiple_terms, strategy_args = list())
```

Arguments

`x`	An object of class `mstATA_design` created by `mst_design()`.
`multiple_terms`	A list of objective terms created by `objective_term()`. The list must contain **two or more** objective terms. All objective terms must be **relative objectives** (i.e., `goal = NULL`).
`strategy_args`	A named list of strategy-specific arguments used by the **capped_maximin** aggregation. Supported fields: • `proportions` – A positive numeric vector of length `length(multiple_terms)` specifying the relative scaling of each objective term. Defaults to a vector of ones.

Details

1. Overview

The capped maximin strategy applies when there are **two or more objective terms**, each representing a linear score of the form

$$y_k = a_k^\top x, k = 1, \ldots, K,$$

where a_k is the coefficient vector constructed by `objective_term()`, and x denotes the binary item-module decision variables.

The goal is to:

- maximize the *minimum normalized performance* across all objective terms, and
- restrict excessive dominance of any single objective via an overflow band.

Unlike the classical maximin strategy (where δ is user-specified or infinite), capped maximin treats δ as a **decision variable** and optimizes it jointly with the common floor.

Each objective term is created using objective_term() and may differ in attribute type, attribute level, and target value.

Key characteristics for each objective term:

(1) Requirement type

- **Categorical**: maximize the number of items from a specified category level; or
- **Quantitative**: maximize the sum of item-level quantitative attributes (e.g., difficulty, discrimination, item information).

(2) Application level

- **"Module-level"** – only items in a specified module contribute;
- **"Pathway-level"** – items in all modules belonging to a pathway;
- **"Panel-level"** – all item selections in the panel contribute.

(3) Objective type

The capped maximin strategy supports **relative objectives only**: maximize $a_k^\top x$

Absolute (goal-based) objectives of the form $|a_k^\top x - g_k|$ are **not permitted** under capped maximin.

2. Proportions and normalization

Each objective term may be associated with a positive proportion $p_k > 0$. These proportions encode the relative scaling of each objective.

By default, $p_k = 1$ for all terms.

3. Overflow control via δ

The capped maximin formulation introduces a common overflow variable $\delta \geq 0$, yielding the constraints:

$$p_k\, y \leq a_k^\top x \leq p_k\, y + \delta, k = 1, \ldots, K.$$

These constraints ensure that:

- all objective terms achieve a highest value proportional to a common normalized floor y, and
- no term exceeds the highest value by more than δ.

Value

A list of class `"compiled_objective"`

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real A sparse constraint matrix (class `"Matrix"`) for continuous decision variables. Must have the same number of rows as `A_binary`.

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary A numeric vector of objective coefficients for binary decision variables.

C_real A numeric vector of objective coefficients for continuous decision variables.

sense Always `"max"` for the capped_maximin strategy.

decisionvar_name_new Character vector indicating the names of new decision variables, same length as `ncol(A_real)`.

decisionvar_type_new A character vector of `"C"` (continuous), same length as `ncol(A_real)`.

notes List of strategy metadata.

Mathematical Formulation

Let x denote the vector of binary item-module decision variables. For each objective term $k = 1, \ldots, K$, define:

$$y_k = a_k^\top x.$$

Introduce two continuous decision variables:

- y – the common normalized floor value;
- δ – the maximum allowable overflow above the floor.

The capped maximin optimization problem is:

$$\max(y - \delta)$$

subject to:

$$p_k\, y \leq y_k \leq p_k\, y + \delta, k = 1, \ldots, K.$$

This formulation encourages balanced improvement across multiple objectives while penalizing excessive disparity among them.

Examples

```
data("mini_itempool")
# MST 1-2-3 design, S1R-S2E-S3H, S1R-S2H-S3E pathways are not allowed.
test_mstATA <- mst_design(
  itempool        = mini_itempool,
  design          = "1-3-3",
  exclude_pathway = c("1-1-3", "1-3-1"),
  pathway_length  = 8
)

# Example: Maximize the minimum TIF across theta = -1, 0, 1
```

```
theta_n1 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=-1)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "max"
)

theta_0 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=0)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "max"
)

theta_1 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=1)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "max"
)


capped_maximin_obj(x = test_mstATA, multiple_terms = list(theta_n1,theta_0,theta_1))
```

`capped_maximin_panel` *Precomputed MST Panel: Capped_Maximin Formulation*

Description

A precomputed multistage testing (MST) panel assembled using the capped_maximin objective formulation described in the *Formulation for Multiple Objectives* vignette.

Usage

```
capped_maximin_panel
```

Format

An object of class `mstATA_panel`.

Details

This object is included to avoid long solver runtimes during vignette building and package checking.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

`check_comblock_feasibility` *Feasibility check for a specified combination of constraint blocks*

Description

Diagnoses infeasibility in an `mstATA_model` by testing whether a user-specified combination of constraint blocks, together with the core set of constraints, results in an infeasible model.

The full model must be infeasible. A reduced model is constructed that always includes the core constraints (objective-related constraints and essential MST structure constraints) and additionally includes the constraint blocks specified in `con_blocks`. The solver status of this reduced model is then evaluated.

Usage

```
check_comblock_feasibility(model_spec, con_blocks, solver, time_limit = 99999)
```

Arguments

`model_spec`	An object of class `mstATA_model`. The full model must be infeasible.
`con_blocks`	An integer vector giving the indices of constraint blocks (rows of `model_spec$specification`) to be jointly tested. These blocks are added to the core constraints as a group.
`solver`	A character string indicating which solver to use. One of `"gurobi"`, `"lpsolve"`, `"HiGHS"`, `"Symphony"`, or `"GLPK"`.
`time_limit`	The maximum amount of computation time allocated to the solver. Default is 99999 seconds.

Details

The reduced model always includes a fixed core set of constraints:

- Objective-related constraints.
- Essential MST structure constraints (e.g., module-/pathway-level item count constraints, routing decision points constraints, item reusage within a panel).

The constraint blocks specified in `con_blocks` are added to this core *simultaneously* and tested as a group. This function does not perform a one-at-a-time or incremental check; instead, it evaluates the feasibility of the combined block set.

This function is primarily intended for second-stage infeasibility diagnostics, such as verifying suspected conflicting constraint combinations or validating results from block-wise (additive or subtractive) diagnostic procedures.

Value

A list with the following elements:

blocks_tested Character vector of requirement names corresponding to `con_blocks`.

status Solver status returned for the reduced model (e.g., `"INFEASIBLE"`, `"OPTIMAL"`, `"FEASIBLE"`).

See Also

`check_singleblock_feasibility()`

---

check_singleblock_feasibility

Block-wise feasibility diagnostics for infeasible mstATA models

---

Description

Diagnoses infeasibility in an `mstATA_model` by solving a sequence of reduced models. The full model must be infeasible. A core set of constraints (objective-related constraints and essential MST structure constraints) is checked for feasibility first. If feasible, the core set of constraints is always included. Each remaining constraint block is then tested individually by solving a reduced model that contains the core constraints plus exactly one additional constraint block (i.e., non-core blocks are not accumulated).

Usage

```
check_singleblock_feasibility(model_spec, solver, time_limit = 99999)
```

Arguments

`model_spec`	An object of class `mstATA_model`. The model must be infeasible.
`solver`	A character string indicating which solver to use. One of `"gurobi"`, `"lpsolve"`, `"HiGHS"`, `"Symphony"`, or `"GLPK"`.
`time_limit`	The maximum amount of computation time allocated to the solver. Default is 99999 seconds.

Details

The core model always includes:

- Objective-related constraints.
- Essential MST structure constraints (e.g., module-/pathway-level item count constraints, routing decision points constraints, item reusage within a panel).

The MST structure constraints ensure that the fundamental structural requirements of the MST design are satisfied.

Specifically, MST structure constraints include:

(1) Item count constraints

(a) Number of items in each module is provided in `mst_design()`

- One constraint is created for each module.
- Each module must contain *exactly* the specified number of items.

(b) Number of items in each pathway is provided in `mst_design()`.

- Each pathway must contain *exactly* the specified number of items.
- Optional stage-level structure constraints may additionally enforce:
  - Minimum number of items per stage.
  - Maximum number of items per stage.
- If `stage_length_bound` = `NULL`, the following default stage-level structure is enforced:
  - Each stage must contain at least one item.
  - Modules within the same stage must have equal length.

(2) Routing decision points constraints

These constraints are generated when `rdps` is provided in `mst_design()`.

- The test information at the routing decision point between adjacent modules in the next stage is similar within a specified tolerance.

(3) Panel-level item usage constraints

These constraints are generated using `panel_itemreuse_con()` limiting how many times an item may be selected across the modules and pathways contained within a single MST panel.

After checking the feasibility of the core set of constraints, all other (non-structural) constraint blocks are tested individually by adding them to the core model one at a time. This function does not attempt to diagnose infeasibility arising from interactions among multiple non-core constraint blocks.

Value

A data frame with one row per constraint block tested.

See Also

`test_itemcount_con()`,

`test_rdp_con()`,

`mst_structure_con()`,

`panel_itemreuse_con()`,

`check_comblock_feasibility()`,

`solve_model()`

compute_icc *Compute Item Characteristic Curves (ICC)*

Description

Computes item category characteristic curves (ICCs) for dichotomous, polytomous, or mixed-format item pools. The function supports multiple IRT models simultaneously and returns category-level response probabilities for each item at specified ability points.

Usage

```
compute_icc(items, item_par_cols, theta, model_col, nrCat_col = NULL, D = 1)
```

Arguments

items	A data frame containing item metadata. Each row represents an item. The data frame must include a column specifying the item model (see model_col) and the parameter columns referenced in item_par_cols.
item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in items[[model_col]]: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names in items that contain the required item parameters for the corresponding model.
theta	A numeric vector of ability values at which the ICCs are evaluated.
model_col	A character string specifying the column name in items that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", or "NRM".
nrCat_col	A character string specifying the column name in items that indicates the number of response categories for each items. Default is NULL, the number of response categories is derived based on item parameters. If not NULL, the number of response categories is checked with number of item parameters for consistency.
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.

Details

The output matrices returned by compute_icc() include a fixed set of category columns (cat0, cat1, ...), with unused categories filled with zeros to ensure consistent dimensions across mixed-format item pools.

(1) **Dichotomous models.**

For the models "1PL", "RASCH", "2PL", "3PL", and "4PL", item parameters are extended to the following four columns:

`discrimination` Slope parameter (a).

`difficulty` Location parameter (b).

`guessing` Lower asymptote (c).

`upper` Upper asymptote (d).

Missing parameters required for a given model are automatically filled. For example, the RASCH and 1PL models are represented with $a = 1$, $c = 0$, and $d = 1$.

All dichotomous item parameters are validated prior to ICC computation:

- `discrimination > 0`
- `0 <= guessing <= 1`
- `0 <= upper <= 1`
- `guessing < upper`

(2) **Polytomous models.**

The supported polytomous models include `"GRM"`, `"MGRM"`, `"PCM"`, `"GPCM"`, `"RSM"`, and `"NRM"`. Let K denotes the number of non-baseline response categories. Required parameters depend on the specific model and may include discrimination parameters and category-specific thresholds or step parameters (e.g., `betaj1`, ..., `betajK`, `deltaj1`, ..., `deltajK`). See Pi_internal().

Value

A named list of matrices, one for each value in `theta`. List length = the number of ability values and List names = Character strings of the form `"theta=<value>"`.

Each matrix entry gives the probability of responding in the corresponding category for the given item at the specified ability level. Matrix rows = the number of items, with row names equal to item identifiers. Matrix columns = Response categories labeled `cat0`, `cat1`, ..., `catM`.

Examples

```
## Example 1: Dichotomous model
data("mini_itempool")
compute_icc(mini_itempool,list("3PL"=c("discrimination","difficulty","guessing")),
            theta = c(-1,0,1),model_col = "model",D = 1.7)

## Example 2: Polytomous model (PCM)
data("poly_itempool")
compute_icc(poly_itempool,list("PCM"=c("deltaj1","deltaj2","deltaj3")),
            theta = c(-1,0,1),model_col = "model")

## Example 3: multiple IRT models (3PL + GPCM + GRM)
data("Rmst_pool")
compute_icc(Rmst_pool,
            item_par_cols = list("3PL"=c("a","b","c"),
                                 "GPCM" = c("alpha","delta1","delta2","delta3"),
                                  "GRM" = c("alpha","beta1","beta2")),
            theta = c(-1,0,1),model_col = "model")
```

compute_iif *Compute Item Information Function (IIF) at Target Theta Points*

Description

Computes item information function (IIF) values for dichotomous, polytomous, or mixed-format item pools across a set of target ability values. The function supports multiple unidimensional IRT models simultaneously and returns an item-by-theta information matrix.

Usage

```
compute_iif(items, item_par_cols, theta, model_col, D = 1)
```

Arguments

items	A data frame containing item metadata. Each row represents an item. The data frame must include a column specifying the item model (see model_col) and the parameter columns referenced in item_par_cols.
item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in items[[model_col]]: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names in items that contain the required item parameters for the corresponding model.
theta	A numeric vector of ability values at which the IIFs are evaluated.
model_col	A character string specifying the column name in items that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", or "NRM".
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.

Value

A numeric matrix of item information values, with rows corresponding to items and columns corresponding to theta.

See Also

compute_icc()

Examples

```
## Example 1: Dichotomous model
data("mini_itempool")
compute_iif(mini_itempool,
            item_par_cols = list("3PL"=c("discrimination","difficulty","guessing")),
```

```
            theta = c(-1,0,1),model_col = "model")

## Example 2: Polytomous model (PCM)
data("poly_itempool")
compute_iif(poly_itempool,
            item_par_cols = list("PCM"=c("deltaj1","deltaj2","deltaj3")),
            theta = c(-1,0,1),model_col = "model")

## Example 3: multiple IRT models (3PL + GPCM + GRM)
data("Rmst_pool")
compute_iif(Rmst_pool,
            item_par_cols = list("3PL"=c("a","b","c"),
                                 "GPCM" = c("alpha","delta1","delta2","delta3"),
                                  "GRM" = c("alpha","beta1","beta2")),
            theta = c(-1,0,1),model_col = "model")
```

`concat_enemy_sets` *Concatenate Enemy Sets*

Description

Concatenates enemy sets produced by `create_enemy_sets()`. Enemy sets from each source are **appended without modification**, while pairwise enemy relationships are **unioned** across all inputs. No transitive closure or set expansion is performed.

Usage

```
concat_enemy_sets(...)
```

Arguments

`...`	One or more enemy relationships returned by `create_enemy_sets()`. Each input must be a list with the following components:
	ExclusionPair A two-column character matrix of pairwise enemy relationships.
	EnemySet A list of character vectors representing explicitly defined enemy sets.

Details

Suppose source A defines five enemy sets and source B defines two enemy sets. The resulting object will contain **seven enemy sets**, formed by simple concatenation of the input `EnemySet` components.

The `ExclusionPair` component of the result is the union of all unique, unordered pairwise enemy relationships across inputs.

Importantly, no transitive inference is applied. If item A is an enemy of B and item B is an enemy of C (possibly from different sources), this function does **not** infer that A is an enemy of C.

This function is intended for situations where enemy relationships are defined from multiple independent sources (e.g., similarity, cluing, or security rules) and must be preserved exactly as specified.

Value

A list with components:

ExclusionPair A two-column character matrix of unioned enemy pairs.

EnemySet A list of enemy sets formed by concatenating all inputs.

See Also

create_enemy_sets

Examples

```
## Example: combining multiple sources of enemy relationships
## Enemy relationships based on similarity
similarity <- create_enemy_sets(
  id_col    = mini_itempool$item_id,
  enemy_col = mini_itempool$enemy_similarity
)

## Enemy relationships based on cluing
cluing <- create_enemy_sets(
  id_col    = mini_itempool$item_id,
  enemy_col = mini_itempool$enemy_cluing
)

## Combine all enemy relationships into a single enemy set
enemy_all <- concat_enemy_sets(
  similarity,
  cluing
)
enemy_all
```

---

create_enemy_sets *Create Enemy Pairs and Enemy Sets*

Description

Create pairwise enemy relationships and corresponding enemy sets for items or stimuli based on metadata in the item pool.

This function parses enemy information from two aligned vectors– an identifier column and an enemy-specification column–and returns (1) a matrix of unique enemy pairs and (2) a list of enemy sets. Enemy pairs represent item or stimulus combinations that must not be selected together. See Details.

The function is intended as a **preprocessing step** prior to constructing enemy-item or enemy-stimulus exclusion constraints.

Usage

```
create_enemy_sets(id_col, enemy_col, sep_pattern = ",")
```

Arguments

id_col	A character vector of item or stimulus identifiers (e.g., item IDs or stimulus names).
enemy_col	A character vector specifying enemy items or stimuli associated with each element of id_col. The length must equal length(id_col). Missing values indicate no enemies.
sep_pattern	A regular expression used to split multiple enemy identifiers within enemy_col. Defaults to ",".

Details

1. Enemy definition

Two items or two stimuli are considered *enemies* if they must not appear together. Typical reasons include:

- High surface similarity or paraphrasing
- Shared clues or solution pathways
- Security, exposure, or operational constraints

The ExclusionPair component is the *authoritative definition* of explicitly declared enemy relationships. Each row represents a single, directly specified pairwise exclusion.

The EnemySet component provides a grouped representation of enemy relationships for constraint construction. Enemy sets may include items or stimuli that are connected through one or more pairwise exclusions in ExclusionPair. Formally, each enemy set corresponds to a connected component of the graph induced by ExclusionPair.

2. Interpretation of enemy relationships

Enemy relationships specified through enemy_col are interpreted *transitively*. That is, if item/stimulus A is an enemy of B, and B is an enemy of C, then A, B, and C are merged into a single *enemy set*. From each enemy set, at most one item or stimulus may be selected.

This conservative interpretation prevents indirect information leakage or similarity chains that may not be obvious from pairwise specifications alone.

3. Important modeling note

If users intend different *types* of enemy relationships that should *not* propagate transitively, they should encode them in *separate columns* of the item pool.

For example:

- Item A is an enemy of item B because they are *too similar*.
- Item B is an enemy of item C because they *provide clues to each other*.
- The user does *not* believe that item A and item C should be treated as enemies.

In this case, a safer modeling strategy is:

- Encode the A-B enemy relationship in one enemy column (e.g., enemy_similarity);
- Encode the B-C enemy relationship in a different enemy column (e.g., enemy_clueing);
- Construct enemy sets separately from each column.

This approach prevents unintended transitive grouping, allowing items A and C to remain eligible for joint selection while still enforcing the intended pairwise exclusions.

In this case, users construct enemy sets separately, concatenate them using concat_enemy_sets() and supply the concatenated result to mst_design().

Value

A list with the following components:

ExclusionPair A two-column character matrix in which each row represents a unique pair of mutually exclusive (enemy) items or stimuli.

EnemySet A list of character vectors, where each element represents a set of items or stimuli that are mutually exclusive with one another.

See Also

`concat_enemy_sets()`

Examples

```
## Example 1: Enemy item relationships
create_enemy_sets(
  id_col = mini_itempool$item_id,
  enemy_col = mini_itempool$enemy_similarity,
  sep_pattern = ","
)

## Example 2: Enemy stimulus relationships
create_enemy_sets(
  id_col = reading_itempool$stimulus,
  enemy_col = reading_itempool$enemy_stimulus,
  sep_pattern = ", "
)
```

---

create_pivot_stimulus_map

Create Pivot-Stimulus Mapping

Description

Construct a stimulus-pivot item map from an item pool containing stimulus identifiers and pivot-item indicators. This function extracts the pivot item for each stimulus and identifies all items associated with that stimulus.

This function is typically used as a **preprocessing** step before applying any constraints related with stimuli and stimulus-based items.

Usage

```
create_pivot_stimulus_map(
  itempool,
  item_id_col = "item_id",
  stimulus,
  pivot_item
)
```

Arguments

itempool	A data.frame containing the item pool.
item_id_col	A string giving the column name in itempool that uniquely identifies items.
stimulus	A string giving the column name in itempool that identifies the stimulus or passage to which each item belongs. Values may be character, or numeric (coerced to character). Items not associated with a stimulus should be coded as NA.
pivot_item	A string giving the column name in itempool. Values must be character and non-missing values identify pivot items. Each stimulus must have exactly one pivot item. Empty strings or whitespace are treated as missing. See Details.

Details

Formally, a pivot item is defined as an item that is selected if and only if its corresponding stimulus is selected. In practice, the pivot item is typically chosen as the item within a set that best represents the stimulus–identified by content experts as having the most representative content or desirable psychometric properties.

Value

A list with the following components:

pivot_item_id A character vector of pivot item IDs, one per stimulus.

stimulus_name A character vector giving the stimulus identifier associated with each pivot item. The order matches pivot_item_id.

stimulus_members A named list. Each element contains the item IDs belonging to a stimulus. Names match stimulus_name.

numItems_stimulus An integer vector giving the number of items associated with each stimulus. This is length(stimulus_members[[i]]).

Examples

```
itempool <- data.frame(
  item_id = 1:7,
  stimulus_id = c("S1","S1","S1","S2","S2", NA, NA),
  is_pivot   = c("P", NA, NA, "P", NA, NA, NA)
)

create_pivot_stimulus_map(
  itempool = itempool,
```

```
  item_id_col = "item_id",
  stimulus = "stimulus_id",
  pivot_item = "is_pivot"
)
```

`dvlink_item_solution` *Defining Solution-level Item Indicator Variables and Generating Constraints for Item Exposure Control Across Multiple Panels*

Description

This function does **not** need to be called by users and is internally used by multipanel_spec(). It is a **structural component** of multi-panel MST assembly.

It has two roles.

- Define the semantics of the solution-level indicator variable s_i: an item is considered *used* if and only if it is selected in at least one module of at least one panel.
- Constrain user-specified **minimum and maximum reuse limits** for each item across all panels. The total number of constraints depends on the number of items in the item pool (for min and max usage, two constraints per item).

Usage

```
dvlink_item_solution(
  x,
  num_panels,
  global_min_use = 0,
  global_max_use = Inf,
  item_min_use = NULL,
  item_max_use = NULL
)
```

Arguments

`x`	An object of class "mstATA_design" created by mst_design().
`num_panels`	Integer. Number of panels to be assembled.
`global_min_use`	Scalar. Minimum number of times any item may be used across panels. Default is 0.
`global_max_use`	Scalar. Maximum number of times any item may be used across panels. Default is `Inf`.
`item_min_use`	Optional data frame with columns `item_id` and `min`, specifying item-specific minimum reuse counts. Default is `NULL`.
`item_max_use`	Optional data frame with columns `item_id` and `max`, specifying item-specific maximum reuse counts. Default is `NULL`.

Details

1. Specification

The constraint enforces:

Each item can be used at least a specified minimum number of times and/or no more than a specified maximum number of times across all panels.

Key characteristics:

- Attribute: *itemset level (item-itself set) logical* (if selected, then increment reuse).
- Constraints are applied at the **"Solution-level"**.

2. Number of times an item can be selected

Global reuse limits are supplied via `global_min_use` and `global_max_use`. These bounds apply to all items unless overridden.

Item-specific reuse limits may be supplied through:

- `item_min_use`: a data frame with columns `item_id` and `min`
- `item_max_use`: a data frame with columns `item_id` and `max`

The total number of constraints generated is always $2 \times P$ where P is the pool size, corresponding to one minimum and one maximum constraint per item.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index. Suppose there are P panels assembled in one solution. Let $p = 1, \ldots, P$ denote the panel index.

Decision variable set 1: item-module-panel selection variables

For each item i_s, $x_{i_s,m,p}$ = 1 if item i_s is selected in module m of panel p. $x_{i_s,m,p}$ = 0 otherwise.

Decision variable set 2: solution-level item indicators

For each item i_s, s_{i_s} = 1 if item i_s is used in any module in any panel. s_{i_s} = 0 otherwise.

Number of times item i is selected in a solution

$$u_i = \sum_p \sum_m x_{i_s,m,p}$$

This constraints enforce:

$$m_i^{\min} s_{i_s} \leq u_i \leq m_i^{\max} s_{i_s}$$

ensuring consistency between panel-level selections and solution-level indicators while enforcing reuse limits.

where:

- $x_{i_s,m,p}$ indicates whether item i_s is selected in module m of panel p;
- s_{i_s} indicates whether item i_s is used anywhere in the solution;
- $m_i^{\min}$ and $m_i^{\max}$ are item-specific reuse bounds.

These constraints are **structural**: they define the meaning of the solution-level indicator variables and also create constraints for the item exposure control across panels.

- If $s_i = 0$, then item i cannot be selected in any module of any panel.
- If $s_i = 1$, the total number of selections of item i must lie within the specified reuse bounds.
- If item i is not selected in any module across panels, then s_i must equal 0.
- If item i is selected in at least one module of any panel, then s_i must equal 1.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool      = mini_itempool,
  design        = "1-3-3",
  module_length = c(6, 4, 4, 4, 3, 3, 3)
)

# Example: Allow each item to be used at most once across all panels
dvlink_item_solution(
  x = test_mstATA,
  num_panels = 2,
  global_max_use = 1
)
```

`enemyitem_exclu_con` *Generate Pathway-Level Constraints to Prevent Enemy Items from Appearing Together*

Description

Constructs linear constraints to ensure that items belonging to the same *enemy set* do **not** appear together within any assembled MST pathway (i.e., any complete test form).

Enemy items are items that:

- give away the answer to one another,
- are too similar in content or structure, or
- should not be seen by the same examinee for security or fairness reasons.

The total number of constraints = num of enemy item sets * num of pathways

Usage

```
enemyitem_exclu_con(x)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.

Details

1. Specification

The enforced constraint is:

At most one item from the same enemy set can be selected in a pathway

- The attribute is a *itemset-level logical grouping*: enemyitem_set defines enemy item groups in the item pool.
- The constraint must be applied at **"Pathway-level"**. Because each pathway represents the full sequence of modules that an examinee could encounter, preventing enemy items from appearing in a pathway ensures that no examinee can see more than one item from the same enemy set.

2. Logical Interpretation

For an enemy item set V_e^{item}:If one item in the set is selected anywhere in the pathway, all other items in the same set must be excluded from that pathway.

This ensures that **at most one** item from each enemy group appears in any assembled test form.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_e^{item} denotes a set of items that are mutually exclusive (enemy) items.

$$\sum_{m \in r} \sum_{i_s \in V_e^{item}} x_{i_s,m} \leq 1.$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.

See Also

`create_enemy_sets()`,

`concat_enemy_sets()`

Examples

```
# Example 1: Two enemy sets: {Item1, Item2, Item3} and {Item4, Item5}
data("mini_itempool")

enemyitem_set<- create_enemy_sets(mini_itempool$item_id,
                                  mini_itempool$enemy_similarity,
                                  sep_pattern = ",")
```

```
test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  module_length = c(4,2,2,2,2,2,2),
  enemyitem_set = enemyitem_set
)

enemyitem_exclu_con(x = test_mstATA)


# Example 2:
## Enemy relationships based on similarity
similarity <- create_enemy_sets(
  id_col    = mini_itempool$item_id,
  enemy_col = mini_itempool$enemy_similarity
)

## Enemy relationships based on cluing
cluing <- create_enemy_sets(
  id_col    = mini_itempool$item_id,
  enemy_col = mini_itempool$enemy_cluing
)

## Combine all enemy relationships into a single enemy set
enemy_all <- concat_enemy_sets(
  similarity,
  cluing
)

## Use the combined enemy set in the test design
test_mstATA2 <- mst_design(
  itempool       = mini_itempool,
  design = "1-3-3",
  module_length = c(4,2,2,2,2,2,2),
  enemyitem_set = enemy_all
)

enemyitem_exclu_con(test_mstATA2)
```

`enemystim_exclu_con` *Generate Pathway-Level Constraints to Prevent Enemy Stimuli from Appearing Together*

Description

Constructs linear constraints to ensure that stimuli belonging to the same *enemy set* do **not** appear together within any MST pathway (i.e., any complete test form).

Enemy stimuli are stimuli that should not be viewed by the same examinee for security or fairness reasons.

The total number of constraints = number of enemy stimulus sets * number of pathways.

Usage

```
enemystim_exclu_con(x)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().

Details

1. Specification

The enforced constraint is:

At most one stimulus from the same enemy set can be selected in a pathway

- The attribute is a *itemset-level logical grouping*: enemystim_set defines enemy stimulus groups in the item pool.
- The constraint must be applied at **"Pathway-level"**. Because each pathway represents the full sequence of modules that an examinee could encounter, preventing enemy stimuli from appearing in a pathway ensures that no examinee can see more than one stimulus from the same enemy set.

2. Logical Interpretation

For an enemy stimulus set V_e^{stim}:If a stimulus in the set is selected anywhere in the pathway, all other stimuli in the same set must be excluded from that pathway.

This ensures that **at most one** stimulus from each enemy group appears in any assembled test form.

Value

An object of S3 class "mstATA_constraint" with named elements:

name A character vector indicating the specifications in each row of A_binary

specification A data.frame summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of A_binary.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_e^{stim} denotes a set of stimuli that are mutually exclusive (enemy).

$$\sum_{m \in r} \sum_{s \in V_e^{stim}} x_{i_s^*, m} \leq 1.$$

Here:

- $x_{i_s^*, m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus s is selected in that module.
- $m \in r$ denote modules belonging to pathway r.

See Also

create_enemy_sets(),

concat_enemy_sets(),

create_pivot_stimulus_map()

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)

# Create enemy-stimulus sets
enemystim_set<-create_enemy_sets(
  reading_itempool$stimulus,
  reading_itempool$enemy_stimulus,
  sep_pattern = ","
)

test_mstATA <- mst_design(
  itempool       = reading_itempool,
  design         = "1-3-3",
  module_length  = c(10,12,12,12,15,15,15),
  exclude_pathway = c("1-1-3", "1-3-1"),
  pivot_stim_map = pivot_stim_map,
  enemystim_set = enemystim_set
)

enemystim_exclu_con(x = test_mstATA)
```

`expected_score` *Expected score*

Description

Computes the expected total score implied by item category probability curves. The expectation can be evaluated either at a grid point (single ICC matrix) or at an arbitrary ability value.

Usage

```
expected_score(icc, target_theta = NULL)
```

Arguments

`icc`	Either a numeric matrix of item category probabilities at a fixed ability level θ (Rows correspond to items and columns correspond to response categories (`cat0`, `cat1`, ...).), or a named list of such matrices evaluated on the ability grid.
`target_theta`	Optional numeric value giving the ability level at which the expected score is to be evaluated. Required when `icc` is a list.

Details

1. How expected score is computed

Consider there are I items. Let the score for item i be a discrete random variable Y_i taking values $\{0, 1, \ldots, m\}$. Conditional on ability θ, item responses are assumed to be locally independent, with category probabilities

$$\Pr(Y_i = k \mid \theta) = p_{ik}(\theta), k = 0, \ldots, m.$$

The expected score for item i at ability level θ is

$$E(Y_i \mid \theta) = \sum_{k=0}^{m} k p_{ik}(\theta).$$

The expected total score is the sum of item-level expectations:

$$E(S \mid \theta) = E(\sum_{i=1}^{I} Y_i \mid \theta) = \sum_{i=1}^{I} E(Y_i \mid \theta),$$

where $S = \sum_{i=1}^{I} Y_i$ denotes the total score.

This function evaluates the above expression numerically using the item category probability matrix produced by compute_icc(). Missing categories (if any) are assumed to have zero probability.

2. ICC input

(1) If `icc` is a matrix, the expected score is computed directly as

$$E(S \mid \theta) = \sum_i \sum_k k p_{ik}(\theta),$$

where p_{ik} denotes the probability that item i produces score k at ability level θ.

(2) If `icc` is a list, the expected score function $E(S \mid \theta)$ is first evaluated on the grid encoded in `names(icc)`, and a continuous approximation is constructed via linear interpolation. The expected score at `target_theta` is then obtained from this interpolated test characteristic curve using `stats::approxfun()`. **The numerical accuracy of the interpolated value depends on the density and range of the ability grid used to construct the ICC list.**

Value

A single numeric value giving the expected total score $E(S \mid \theta)$ at the specified ability level.

See Also

`compute_icc()`,

`inverse_tcc()`,

`stats::approxfun()`

Examples

```
## Example 1: Dichotomous model
data("mini_itempool")
icc_mat<-compute_icc(mini_itempool,
                     list("3PL"=c("discrimination","difficulty","guessing")),
                     theta = 0,model_col = "model",D = 1.7)

icc_list<-compute_icc(mini_itempool,
                      list("3PL"=c("discrimination","difficulty","guessing")),
                      theta = seq(-5,5,0.1),model_col = "model",D = 1.7)

expected_score(icc_mat[[1]])
expected_score(icc_list,target_theta=seq(-5,5,0.1))

## Example 2: Polytomous model (PCM)
data("poly_itempool")
icc_mat<-compute_icc(poly_itempool,
                    list("PCM"=c("deltaj1","deltaj2","deltaj3")),
                     theta = 0,model_col = "model")
icc_list<-compute_icc(poly_itempool,
                      list("PCM"=c("deltaj1","deltaj2","deltaj3")),
                      theta = seq(-5,5,0.1),model_col = "model")
expected_score(icc_mat[[1]])
expected_score(icc_list,target_theta=0)

## Example 3: Mixed item format (3PL + GPCM + GRM)
data("Rmst_pool")
icc_mat<-compute_icc(Rmst_pool,
```

```
                      item_par_cols = list("3PL"=c("a","b","c"),
                                           "GPCM" = c("alpha","delta1","delta2","delta3"),
                                           "GRM" = c("alpha","beta1","beta2")),
                      theta = 0,model_col = "model")
icc_list<-compute_icc(Rmst_pool,
                       item_par_cols = list("3PL"=c("a","b","c"),
                                           "GPCM" = c("alpha","delta1","delta2","delta3"),
                                             "GRM" = c("alpha","beta1","beta2")),
                       theta = seq(-5,5,0.1),model_col = "model")
expected_score(icc_mat[[1]])
expected_score(icc_list,target_theta=0)
```

`gen_weight` *Generate population weights over a specified ability grid*

Description

Computes normalized weights on a user-specified ability grid `theta` based on a chosen population distribution. The function evaluates the specified density at each grid point and rescales the resulting values so that the weights sum to 1. The output provides a discrete approximation of the population ability distribution and can be used for marginal summaries in analytic classification accuracy or precision evaluation.

Supports:

- Named base-R distributions via their densities (e.g., `norm`, `t`, `logis`, `unif`, `beta`, `gamma`, `lnorm`,...).
- Finite **mixtures** of named distributions
- **Empirical** distributions via Gaussian-kernel KDE (kernel density estimation)

Weights are normalized so that `sum(w) == 1`. Use them to approximate expectations:

$$\mathbb{E}[f(\theta)] \approx \sum_i w_i f(\theta_i).$$

Usage

```
gen_weight(
  theta = seq(-5, 5, 0.1),
  dist = "norm",
  params = list(mean = 0, sd = 1),
  components = NULL,
  empirical_theta = NULL
)
```

Arguments

theta	Optional numeric vector specifying the ability grid at which the population distribution is evaluated. Default is seq(-5,5,0.1).
dist	Character string specifying the distribution family: one of norm, t, logis, unif, beta, gamma, lnorm, mixture, empirical. Default is norm.
params	List of parameters for the named distribution (when dist is not "mixture", "empirical"). For example, list(mean = 0, sd = 1) for "norm", list(df = 5) for "t", list(location = 0, scale = 1) for "logis". Default is list(mean = 0, sd = 1).
components	For dist = "mixture": a list of components, each like list(dist = "norm", params = list(mean = 0, sd = 1), weight = 0.7). Component weights are rescaled to sum to 1 if needed.
empirical_theta	For dist = "empirical": numeric sample theta values used to build a KDE.

Value

A dataframe with:

theta The supplied ability grid.

w Normalized weights, sum(w) = 1.

Examples

```
## Standard normal population on a custom grid
theta <- seq(-3, 3, by = 0.2)
gen_weight(theta,dist = "norm")

## Normal population with custom parameters
gen_weight(
  theta,
  dist = "norm",
  params = list(mean = -0.5, sd = 1.2)
)

## Empirical population via KDE
gen_weight(
  theta,
  dist = "empirical",
  empirical_theta = rnorm(1000)
)

## Two-component mixture distribution
gen_weight(
  theta,
  dist = "mixture",
  components = list(
    list(dist = "norm", params = list(mean = -1, sd = 0.8), weight = 0.4),
    list(dist = "norm", params = list(mean =  1, sd = 0.8), weight = 0.6)
  )
```

```
)
```

`get_attribute_val` *Get Item Categorical/Quantitative Attribute Value*

Description

Create a vector representing the attribute value.

Usage

```
get_attribute_val(itempool, attribute, cat_level = NULL)
```

Arguments

`itempool`	A `data.frame` containing the item pool.
`attribute`	A string giving the column name in `itempool` that represents the categorical/quantitative attribute.
`cat_level`	A optional character string for one category for the categorical attribute. Default is NULL, indicating the attribute is a quantitative attribute.

Value

A vector of item attribute values. The same length as the item pool.

- If the attribute is a quantitative attribute, return a numeric vector. NA is not allowed in the quantitative attribute.
- If the attribute is a categorical attribute, return a binary integer vector with 1 indicating an item having the categorical attribute, 0 indicating an item not having the categorical attribute. If an item has NA in the categorical attribute, it will be coded as 0, indicating this item not having the categorical attribute.

Examples

```
# Example 1, item categorical attribute
data("mini_itempool")
get_attribute_val(itempool = mini_itempool,attribute = "itemtype",cat_level = "MC")
# Example 2, item quantitative attribute
get_attribute_val(itempool = mini_itempool,attribute = "difficulty",cat_level = NULL)
# Example 3: stimulus categorical attribute
data("reading_itempool")
get_attribute_val(itempool = reading_itempool[!is.na(reading_itempool$pivot_item),],
                  attribute = "stimulus_type",cat_level = "history")
# Example 4: stimulus quantitative attribute
get_attribute_val(itempool = reading_itempool[!is.na(reading_itempool$pivot_item),],
                  attribute = "stimulus_words")
```

goal_programming_obj *Minimize Deviations from Target Goals (Goal Programming)*

Description

Implements a **goal programming** objective that minimizes deviations between achieved objective values and user-specified target goals.

Multiple absolute objective terms are combined into a single optimization problem. Each term contributes one (`mode = "one_dev"`) or two (`mode = "two_dev"`) deviation variables, and the solver minimizes a (weighted) sum of these deviations.

Weights may be assigned to individual objective terms to reflect their relative importance.

Usage

```
goal_programming_obj(x, multiple_terms, strategy_args = list())
```

Arguments

`x`	An object of class `mstATA_design` created by `mst_design()`.
`multiple_terms`	A list of objective terms created by `objective_term()`. The list must contain **two or more** objective terms. All objective terms must be **absolute (goal-based) objectives**, meaning that a finite target value `goal` is specified for each term.
`strategy_args`	A named list of strategy-specific arguments used by the **goal_programming** aggregation. Supported fields: • `mode` – Character string specifying the deviation formulation. Must be either `"one_dev"` (single absolute deviation) or `"two_dev"` (separate positive and negative deviations). • `weights` – Optional positive numeric vector of length(multiple_term) specifying the relative importance of each objective term. Defaults to a vector of ones.

Details

1. Overview

Goal programming applies when there are **two or more objective terms**, each formulated as an **absolute deviation objective** of the form

$$|a_k^\top x - g_k|,$$

where:

- $a_k^\top x$ is the linear score defined by the objective term;
- g_k is the target value specified by the test developer.

Each objective term is created using objective_term() and may differ in attribute type, attribute level, and target value.

Key characteristics for each objective term:

(1) Requirement type

- **Categorical**: minimize deviation from a target number of items belonging to a specified category level; or
- **Quantitative**: minimize deviation from a target sum of item-level quantitative attributes (e.g., difficulty, discrimination, or item information).

(2) Application level

- **"Module-level"** – only items in a specified module contribute;
- **"Pathway-level"** – items in all modules belonging to a pathway;
- **"Panel-level"** – all item selections in the panel contribute.

(3) Objective type

The goal_programming strategy supports **absolute objectives only**.

2. Deviation Modes

Two deviation formulations are supported:

- **One-deviation mode** (`mode = "one_dev"`):
  Introduces one nonnegative deviation variable $d_k \geq 0$ for each term:

  $$|a_k^\top x - g_k| \leq d_k.$$

  The objective minimizes:

  $$\min \sum_{k=1}^{K} w_k d_k.$$

- **Two-deviation mode** (`mode = "two_dev"`):
  Introduces separate positive and negative deviation variables $d_k^+ \geq 0$ and $d_k^- \geq 0$:

  $$a_k^\top x - g_k = d_k^+ - d_k^-.$$

  with constraints:

  $$a_k^\top x - g_k \leq d_k^+$$

  $$g_k - a_k^\top x \leq d_k^-$$

  The objective minimizes:

  $$\min \sum_{k=1}^{K} w_k (d_k^+ + d_k^-).$$

Interpretation:

- d_k^+ measures how much objective k exceeds its target;
- d_k^- measures how much objective k falls short;
- d_k (one-dev) measures total absolute deviation.

Value

An object of S3 class `"compiled_objective"` with named elements:

name A string indicating the objective function name

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real A sparse constraint matrix (class `"Matrix"`) for continuous decision variables. Must have the same number of rows as `A_binary`.

C_binary A numeric vector of objective coefficients for binary decision variables.

C_real A numeric vector of objective coefficients for continuous decision variables.

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

sense Always `"min"` for the goal_programming strategy.

decisionvar_type_new A character vector of `"B"` (binary), same length as `ncol(A_real)`.

decisionvar_name_new Character vector indicating the names of decision variables, same length as `ncol(A_real)`.

notes information about "strategy" (`"goal_programming"`) and "strategy_args" (`"mode"`,`"weights"`)

Mathematical Formulation

For each objective term $k = 1, \ldots, K$:

$$y_k = a_k^\top x,$$

where:

- a_k is the coefficient vector defined by the attribute, level, and application scope;
- x is the vector of binary item-module-panel decision variables;
- g_k is the target value.

Auxiliary deviation variables are introduced to linearize $|a_k^\top x - g_k|$, and the solver minimizes a weighted sum of these deviations.

All goal programming problems are compiled as **minimization** problems.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool        = mini_itempool,
  design          = "1-3-3",
  exclude_pathway = c("1-1-3", "1-3-1"),
  pathway_length  = 8
)
```

```
obj1 <- objective_term(
  test_mstATA, "iif(theta=-1)", NULL,
  "Module-level", which_module = 1,
  sense = "min", goal = 12
)

obj2 <- objective_term(
  test_mstATA, "iif(theta=0)", NULL,
  "Module-level", which_module = 1,
  sense = "min", goal = 15
)

obj3 <- objective_term(
  test_mstATA, "iif(theta=1)", NULL,
  "Module-level", which_module = 1,
  sense = "min", goal = 12
)

# Example 1: One deviation per goal, equal weights
goal_programming_obj(
  x = test_mstATA,
  multiple_terms = list(obj1, obj2, obj3),
  strategy_args = list(mode = "one_dev",weights = NULL))

# Example 2: One deviation per goal, unequal weights
goal_programming_obj(
  x = test_mstATA,
  multiple_terms =  list(obj1, obj2, obj3),
  strategy_args = list(mode = "one_dev",weights = c(1,0.5,1)))

# Example 3: Two deviations per goal
goal_programming_obj(
  x = test_mstATA,
  multiple_terms = list(obj1, obj2, obj3),
  strategy_args = list(mode = "two_dev",weights = NULL))
```

`goal_programming_panel` *Precomputed MST Panel: Goal_Programming Formulation*

Description

A precomputed multistage testing (MST) panel assembled using the goal_programming objective formulation (one_dev) described in the *Formulation for Multiple Objectives* vignette.

Usage

```
goal_programming_panel
```

Format

An object of class `mstATA_panel`.

Details

This object is included to avoid long solver runtimes during vignette building and package checking.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

Hy13_panel *Precomputed Hybrid MST Assembly Example*

Description

A precomputed MST panel assembled using the hybrid strategy demonstrated in the *Three ATA Strategies* Vignette.

Usage

```
Hy13_panel
```

Format

An object of class `mstATA_panel`.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

---

inverse_tcc *Inverse test characteristic curve*

Description

Computes the inverse Test Characteristic Curve (ITCC) by mapping each possible observed total score to its corresponding estimated ability value (θ). The inversion is performed numerically using a dense theta grid and root-finding.

The function supports mixed-format tests, including dichotomous (e.g., 1PL, 2PL, 3PL, 4PL) and polytomous models (e.g., GRM, PCM, GPCM, RSM, NRM), provided the required item parameters are correctly specified in `item_par_cols`.

For models with guessing (e.g., 3PL/4PL), scores below the lower asymptote are handled using linear interpolation toward the lower bound specified by `range_tcc`.

Usage

```
inverse_tcc(
  items,
  item_par_cols,
  model_col,
  D = 1,
  range_tcc = c(-5, 5),
  tol = 1e-04
)
```

Arguments

items	A data frame containing item metadata. Each row represents an item. The data frame must include a column specifying the item model (see model_col) and the parameter columns referenced in item_par_cols.
item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in items[[model_col]]: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names in items that contain the required item parameters for the corresponding model.
model_col	A character string specifying the column name in items that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", or "NRM".
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.
range_tcc	A numeric vector of length two specifying the lower and upper bounds of the theta search interval. The estimated theta values are constrained within this range. Defaults to c(-5, 5).
tol	A numeric tolerance passed to the root-finding algorithm. Smaller values yield more precise solutions at increased computational cost. Default is 1e-4.

Value

A data frame with two columns:

sum.score All possible observed total scores.

est.theta Estimated theta corresponding to each total score.

Mathematical Formulation

Consider I items are taken by an examinee. Let the score for item i be a discrete random variable Y_i taking values $\{0, 1, \ldots, m\}$, where m is the maximum item score for item i. Conditional on ability θ, item responses are assumed to be locally independent, with category probabilities

$$\Pr(Y_i = k \mid \theta) = p_{ik}(\theta), k = 0, \ldots, m.$$

The expected score for item i at ability level θ is

$$E(Y_i \mid \theta) = \sum_{k=0}^{m} k p_{ik}(\theta),$$

and the expected total module score (the test characteristic curve) is

$$T(\theta) = E(S \mid \theta) = \sum_{i=1}^{I} E(Y_i \mid \theta),$$

where $S = \sum_{i=1}^{I} Y_i$ denotes the total module score.

Given a target score s^*, the inverse TCC problem is to find $\hat{\theta}$ such that

$$T(\hat{\theta}) = s^*.$$

Because $T(\theta)$ generally has no closed-form inverse, this function evaluates $T(\theta)$ on a grid of ability values using ICCs. A root-finding algorithm is then used to solve

$$T(\theta) - s^* = 0$$

on a specified ability interval.

3PL and Non-Invertible Scores

When items follow the three-parameter logistic (3PL) model with guessing parameters $g_i > 0$, the TCC has a nonzero lower bound:

$$\lim_{\theta \to -\infty} TCC(\theta) = \sum_{i=1}^{I} g_i = G.$$

Therefore, any NC score $S < G$ cannot be obtained from the TCC for any finite θ, and the inverse TCC is undefined in the strict sense.

To ensure a monotone score-to-theta mapping, the minimum invertible NC score is defined as

$$X = \lceil G \rceil,$$

the smallest integer strictly greater than the guessing floor.

Let $\theta_{\min}$ denote the lower bound of the restricted ability range (given by `range_tcc[1]`), and let θ_X denote the inverse-TCC solution corresponding to score X.

For any score Y such that $0 \leq Y < X$, the ability estimate is obtained by linear interpolation between $(0, \theta_{\min}$ and (X, θ_X):

$$\theta^*(Y) = \theta_{\min} + \frac{Y}{X}(\theta_X - \theta_{\min}).$$

This construction:

- Preserves monotonicity of the score-to-theta function;

- Respects the lower bound imposed by guessing;
- Ensures continuity at $Y = X$;
- Avoids artificial extrapolation of the TCC.

Scores above the maximum attainable TCC value are mapped to the upper bound `range_tcc[2]`.

Examples

```
# Example 1: dichotomous items (guessing is not allowed)
items<-data.frame(discrimination = c(1,1),difficulty = c(-1,0),
                  model= rep("2PL",2))
inverse_tcc(items = items,
                 item_par_cols = list("2PL"=c("discrimination","difficulty")),
                 model_col = "model",range_tcc = c(-5,5))

# Example 2: dichotomous items, (guessing is allowed)
items<-data.frame(discrimination = c(1,1),difficulty = c(-1,0),guessing = c(0.2, 0.2),
                  model= rep("3PL",2))
inverse_tcc(items = items,
                item_par_cols = list("3PL"=c("discrimination","difficulty","guessing")),
                 model_col = "model", range_tcc = c(-5,5))

# Example 3: polytomous items
items <- data.frame(alphaj = c(1.169,1.052,0.828,0.892,0.965),
                    betaj1 = c(-0.006,-2.405,-0.799,-1.148,-0.299),
                    betaj2 = c(0.487,1.125,0.764,-0.289,1.512),
                    model = rep("GRM",5))
inverse_tcc(items = items,
                 item_par_cols = list("GRM"=c("alphaj","betaj1","betaj2")),
                 model_col = "model", range_tcc = c(-5,5))

# Example 4: dichotomous and polytomous items
items <- data.frame(discrimination = c(1,1,rep(NA,5)),
                    difficulty = c(-1,0,rep(NA,5)),
                    alphaj = c(rep(NA,2),1.169,1.052,0.828,0.892,0.965),
                    betaj1 = c(rep(NA,2),-0.006,-2.405,-0.799,-1.148,-0.299),
                    betaj2 = c(rep(NA,2),0.487,1.125,0.764,-0.289,1.512),
                    model = c(rep("2PL",2),rep("GRM",5)))
inverse_tcc(items = items,
                 item_par_cols = list("2PL"=c("discrimination","difficulty"),
                                      "GRM"=c("alphaj","betaj1","betaj2")),
                 model_col = "model", range_tcc = c(-5,5))
```

`itemcat_con` *Generate Item-Level Constraints to Explicitly Select or Not Select Specific Items*

Description

This function generates linear constraints that require specific items to be either must be **selected** or **not selected** in a multistage test (MST) panel assembly.

If `item_module_eligibility` is supplied in `mst_design()` and the requested item selection conflicts with module eligibility restrictions, an error is thrown.

For details about `item_module_eligibility`, see mst_design().

Constraints may be applied at the `"Module-level"`, `"Pathway-level"`, or `"Panel-level"`.

If `item_module_eligibility` is `NULL`, the number of constraints is:

- **Module-level:** when `which_module` is provided and `which_pathway = NULL`
  The number of constraints is **(number of items in item_ids) x (number of specified modules)**
- **Pathway-level:** when `which_pathway` is provided and `which_module = NULL`
  The number of constraints is **(number of items in item_ids) x (number of specified pathways)**
- **Panel-level:** both `which_module` and `which_pathway` are `NULL`
  The number of constraints is **(number of items in item_ids)**

Usage

```
itemcat_con(
  x,
  item_ids,
  select = TRUE,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`item_ids`	A numeric vector of item row indices, or a character vector of item IDs from `x$ItemPool`.
`select`	Logical. Indicates whether items in `item_ids` must be `TRUE` (selected) or `FALSE` (not selected). Default = `TRUE`.
`which_module`	Optional integer vector of module indices to which the constraints apply.
`which_pathway`	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforced by this function is:

Items listed in `item_ids` must (or must not) be selected.

Key characteristics:

- The attribute type is categorical (each item has a unique ID).
- The attribute is defined at the item-level in the item pool.
- Application levels:
  - **Module-level**: enforce selection in specific module(s).
  - **Pathway-level**: enforce selection in any module of a pathway.
  - **Panel-level**: enforce selection somewhere in the panel.

2. Examples of Interpretation

- **Item i must be selected in the routing module** (Module-level): `item_ids = i`, `which_module = 1`
- **Item i must be selected in Stage 2 easy and medium modules** (modules not belonging to the same pathway) (Module-level): `which_module = c(2,3)`
- **Item i must be selected in the RM pathway** (Pathway-level): `which_pathway = 2`
- **Item i must appear somewhere in the assembled panel** (Panel-level): `which_module = NULL`, `which_pathway = NULL`

3. Interaction with `panel_itemreuse_con()`

When `overlap = FALSE` in `panel_itemreuse_con()`, each item can at most be selected once in a panel.

Therefore:

- `which_module` or `which_pathway` must be a scalar (not a vector).
- If both are `NULL`, the constraint simply enforces that the item must appear somewhere in the panel, without specifying the module/pathway.

When `overlap = TRUE` in `panel_itemreuse_con()`, each item can at most be selected once in a pathway, but each item may be selected in multiple modules within the same stage.

Therefore:

- `which_module` may be a vector (can not be modules that appear in the same pathway)

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A right-hand-side vector of 1s or 0s depending on whether `select` is `TRUE` or `FALSE`.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

1. Module-level item selection/not selection

In specified module m: for all i_s specified in `item_ids`

$x_{i_s,m} = 1$ if item i_s must be selected in module m.

$x_{i_s,m} = 0$ if item i_s must not be selected in module m.

2. Pathway-level item selection/not selection

In specified pathway r: for all i_s specified in `item_ids`

$\sum_{m \in r} x_{i_s,m} = 1$ if item i_s must be selected in pathway r.

$\sum_{m \in r} x_{i_s,m} = 0$ if item i_s must not be selected in pathway r.

3. Panel-level item selection/not selection

In a panel: for all i_s specified in `item_ids`

$\sum_{m} x_{i_s,m} = 1$ if item i_s must be selected in a panel.

$\sum_{m} x_{i_s,m} = 0$ if item i_s must not be selected in a panel.

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.

See Also

`mst_design()`,

`panel_itemreuse_con()`

Examples

```
data("mini_itempool")
test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  module_length = c(4,2,2,2,2,2,2)
)

# Example 1: Item 2 must be selected in the routing module
itemcat_con(x = test_mstATA, item_ids = 2, select = TRUE, which_module = 1)

# Example 2: Item 2 must appear somewhere in the MST panel
itemcat_con(x = test_mstATA, item_ids = 2, select = TRUE)

# Example 3: Item 2 must be selected in Stage 2 easy and medium modules
itemcat_con(x = test_mstATA, item_ids = 2, select = TRUE, which_module = c(2,3))
```

```
# Example 4: Item 2 must be selected in the REE pathway
itemcat_con(x = test_mstATA, item_ids = 2, select = TRUE, which_pathway = 1)

# Example 5: Items 1 and 3 must be selected in the routing module
itemcat_con(x = test_mstATA, item_ids = c(1,3), select = TRUE, which_module = 1)

# Example 6: Items 1 and 3 must be selected in the REE pathway
itemcat_con(x = test_mstATA, item_ids = c(1,3), select = TRUE, which_pathway = 1)

# Example 7: Items 1 and 3 must appear somewhere in the panel
itemcat_con(x = test_mstATA, item_ids = c(1,3), select = TRUE)


# Example 8: With item_module_eligibility provided
mini_itempool <- mini_itempool[order(mini_itempool$difficulty), ]
item_module_eligibility <- list(
  `2` = 1:15,
  `3` = 11:25,
  `4` = 15:30
)

new_test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  module_length = c(6,4,4,4,3,3,3),
  item_module_eligibility = item_module_eligibility
)
itemcat_con(new_test_mstATA, item_ids = 2, select = TRUE, which_module = 2)

# This call produces an error because item 2 is not eligible for module 3
try(itemcat_con(new_test_mstATA, item_ids = 2, select = TRUE, which_module = c(2,3)))
```

`itemquant_con` *Generate Item-Level Constraints Requiring Quantitative Attributes to Satisfy Lower, Upper, or Range Bounds (Not for Operational Use)*

Description

This function constructs linear constraints enforcing that every selected item satisfies a quantitative attribute requirement (e.g., difficulty greater than c, item length less than c, discrimination within a range).

Constraints may be applied at the `"Module-level"`, `"Pathway-level"`, or `"Panel-level"`.

This function is for demonstration only and is *not intended for operational use*. It generates an excessively large number of rows in the constraint matrix and requires item-level attributes to be strictly positive.

The number of generated linear constraints depends on the application level:

- **Module-level:** when `which_module` is provided
  The number of constraints is **(number of items) x (number of modules specified) x side**
- **Pathway-level:** when `which_pathway` is provided and `which_module = NULL`
  The number of constraints is **(number of items) x (number of unique modules in specified pathways) x side**
- **Panel-level:** both `which_module` and `which_pathway` are `NULL`
  The number of constraints is **(number of items) x (total number of modules) x side**

where `side = 1` for one-sided constraints (min-only or max-only) and `side = 2` when both `min` and `max` are provided and min is not equal to max.

Usage

```
itemquant_con(
  x,
  attribute,
  min = NULL,
  max = NULL,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that represents the **item-level quantitative attribute**.
`min`	A numeric scalar for the lower bound.
`max`	A numeric scalar for the upper bound.
`which_module`	Optional integer vector of module indices to which the constraints apply.
`which_pathway`	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforces:

Each selected item must satisfy the specified quantitative lower bound, upper bound, or both.

Key characteristics:

- The attribute type is *quantitative*.
- The attribute is defined at the *item level* in the item pool.
- Application levels:
  - **Module-level**: enforce items selected in specific module(s) satisfy the quantitative requirement.

- **Pathway-level**: enforce items selected in any module of a pathway satisfy the quantitative requirement.
- **Panel-level**: enforce items selected in the panel satisfy the quantitative requirement.

2. Important Restriction

There are two reasons this approach is discouraged. First, item-level quantitative constraints generate an excessively large number of constraint rows, leading to poor computational efficiency. Second, the formulation requires scaling item-selection variables by the attribute value q_{i_s}. If q_{i_s} is zero or negative, the resulting inequalities become invalid.

Because many commonly used item attributes (e.g., IRT difficulty parameters or centered text indices) may take zero or negative values, **this function is not suitable for operational MST assembly**.

Instead, per-item quantitative rules should be enforced during the design stage using `item_module_eligibility()` within mst_design(), which excludes ineligible items prior to module assembly.

Value

This function always throws an error indicating that it is **not for operational use**.

It is retained solely as an illustrative example of how item-level quantitative constraints propagate through different hierarchical levels of an MST design.

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let q_{i_s} be the quantitative attribute value of item i_s.

Upper-bound constraint:

$$q_{i_s}\, x_{i_s,m} \leq b_q^{item,max}$$

Lower-bound constraint:

$$b_q^{item,min}\, x_{i_s,m} \leq q_{i_s}$$

For two-sided range constraints, both inequalities are included. (`min` and `max` both provided and not equal), the number of constraints is doubled (`side = 2`). For one-sided constraints, `side = 1`.

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- q_{i_s} (**must be positive values**) denote the values of a quantitative attribute for item i_s.
- $b_q^{item,min}$ and $b_q^{item,max}$ are the lower and upper allowable bounds for the attribute q_{i_s}.

See Also

`mst_design()`

`joint_module_score_dist` *Compute joint/cumulative score distribution under routing*

Description

Computes the joint (cumulative) score distribution after administering a next-stage module, conditional on routing into a branch defined by a set of reachable previous scores.

This function performs a *convolution* between:

- the conditional score distribution from previous stages, and
- the score distribution of the next module.

The returned distribution is **not normalized**: for each theta column, the column sum equals the probability of routing into this branch at that theta, which does not necessarily have to be 1.

Usage

```
joint_module_score_dist(
  cdist_by_prev,
  prev_scores,
  icc_by_next_mod,
  possible_joint_score
)
```

Arguments

`cdist_by_prev`	Numeric matrix. Rows correspond to previous score values given in `prev_scores`; columns correspond to theta points.
`prev_scores`	Integer vector of non-negative values giving the score values associated with the rows of `cdist_by_prev`.
`icc_by_next_mod`	Named list of ICC matrices for the next module. Each element corresponds to a theta point. Names must match the column names of `cdist_by_prev`.
`possible_joint_score`	Integer vector of non-negative cumulative scores that are reachable under this routing branch.

Details

Let S_{prev} denote the cumulative score prior to the current stage, and S_{next} the score from the next-stage module. This function computes:

$$P(S = s \mid \theta) = \sum_k P(S_{prev} = k \mid \theta)\, P(S_{next} = s - k \mid \theta)$$

for all s in `possible_joint_score`.

Value

A numeric matrix with:

- rows corresponding exactly to `possible_joint_score`,
- columns corresponding to theta points.

Each column represents the (unnormalized) joint score distribution at that theta.

Examples

```
## toy previous score distribution
prev_scores <- 0:2
cdist_by_prev <- matrix(
  c(0.2, 0.3,
    0.5, 0.4,
    0.3, 0.3),
  nrow = 3,
  dimnames = list(NULL, c("theta=-1", "theta=0"))
)

## simple ICC: two items, dichotomous
icc <- matrix(
  c(0.4, 0.6,
    0.7, 0.3),
  nrow = 2,
  byrow = TRUE
)
colnames(icc) <- c("0", "1")

icc_by_next_mod <- list(
  "theta=-1" = icc,
  "theta=0"  = icc
)

possible_joint_score <- c(1, 2, 3)

joint_module_score_dist(
  cdist_by_prev,
  prev_scores,
  icc_by_next_mod,
  possible_joint_score
)
```

`maximin_obj` *Maximin objective (with optional overflow band)*

Description

Implements a **maximin** objective that maximizes a common floor value y across K linear objective scores $u_k = a_k^\top x$.

The compiled constraints enforce:

$$a_k^\top x \geq p_k y, k = 1, \ldots, K,$$

and, if an overflow bound `delta` is finite for term k,

$$a_k^\top x \leq p_k y + \delta_k.$$

Usage

```
maximin_obj(x, multiple_terms, strategy_args = list())
```

Arguments

`x`	An object of class `mstATA_design` created by `mst_design()`.
`multiple_terms`	A list of objective terms created by `objective_term()`. The list must contain **two or more** objective terms. All objective terms must be **relative objectives** (i.e., `goal = NULL`).
`strategy_args`	A named list of strategy-specific arguments used by the **maximin** aggregation. Supported fields: • `proportions` – A positive numeric vector of length(multiple_terms) specifying the relative scaling of each objective term. Defaults to a vector of ones. • `delta` – A nonnegative numeric value controlling the overflow band. May be either a scalar or a numeric vector of length(multiple_terms). Each element must be strictly positive or `Inf`. Defaults to `Inf` (no overflow control).

Details

1. Overview

The maximin strategy applies when there are **two or more objective terms**. Each objective term represents a linear score of the form

$$y_k = a_k^\top x,$$

constructed by objective_term().

The goal is to maximize the minimum (normalized) score across all terms, ensuring balanced performance rather than optimizing any single criterion.

Key characteristics for each objective term:

(1) Attribute type

Each objective term may be defined using either:

- a **categorical attribute** – maximizing or minimizing the number of items belonging to a specific category level; or
- a **quantitative attribute** – maximizing or minimizing the sum of item-level attribute values (e.g., difficulty, discrimination, item information function values).

(2) Application level

Objective terms may be applied at one of three hierarchical levels:

- **"Module-level"** – only items in a specified module contribute;
- **"Pathway-level"** – items in all modules belonging to one pathway;
- **"Panel-level"** – all item selections in the panel contribute.

(3) Objective type

The maximin strategy supports **relative objectives only**:

- **Relative objective**: maximize $a_k^\top x$.

Absolute (goal-based) objectives of the form $|a_k^\top x - g_k|$ are not permitted under the maximin strategy.

2. Proportions and normalization

Each objective term may be associated with a positive proportion p_k. These proportions encode the relative scaling of each objective.

By default, $p_k = 1$ for all terms.

3. Overflow control via `delta`

Two formulations are supported depending on the value of `delta`:

- **Without overflow control** (`delta = Inf`):

$$a_k^\top x \geq p_k y, k = 1, \ldots, K.$$

  This is the classical maximin formulation, enforcing only a lower bound on each objective score.

- **With overflow control** (`delta < Inf`):

$$p_k y \leq a_k^\top x \leq p_k y + \delta_k, k = 1, \ldots, K.$$

  This bounded formulation limits how far any single objective score may exceed the common floor, improving balance and numerical stability.

Value

A list of class `"compiled_objective"`

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` including "Requirement","Attribute","Type","Application Level","Operator","Num of Constraints"

A_binary A sparse constraint matrix (class `"Matrix"`) for binary decision variables. Rows correspond to constraints; columns correspond to binary decision variables.

A_real A sparse constraint matrix (class `"Matrix"`) for continuous decision variables. Must have the same number of rows as `A_binary`.

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary A numeric vector of objective coefficients for binary decision variables.

C_real A numeric vector of objective coefficients for continuous decision variables.

sense Always `"max"` for the maximin strategy.

decisionvar_name_new Character vector indicating the names of new decision variables, same length as `ncol(A_real)`.

decisionvar_type_new A character vector of `"C"` (continuous), same length as `ncol(A_real)`.

notes List of metadata.

Mathematical Formulation

Let x denote the vector of binary item-module-panel decision variables. For each objective term $k = 1, \ldots, K$, define:

$$y_k = a_k^\top x.$$

Introduce a continuous auxiliary variable y representing the minimum normalized score. The maximin optimization problem is:

$$\max y$$

subject to:

$$y_k \geq p_k y, k = 1, \ldots, K,$$

and optionally,

$$y_k \leq p_k y + \delta_k, k = 1, \ldots, K.$$

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool        = mini_itempool,
  design          = "1-3-3",
  exclude_pathway = c("1-1-3", "1-3-1"),
  pathway_length  = 8
)

# Example 1: Maximize the minimum TIF across theta = -1, 0, 1
```

```
theta_n1 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=-1)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "max"
)

theta_0 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=0)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "max"
)

theta_1 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=1)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "max"
)


maximin_obj(x = test_mstATA, multiple_terms = list(theta_n1, theta_0, theta_1))
```

`maximin_panel` *Precomputed MST Panel: Maximin Formulation*

Description

A precomputed multistage testing (MST) panel assembled using the maximin objective formulation described in the *Formulation for Multiple Objectives* vignette.

Usage

```
maximin_panel
```

Format

An object of class `mstATA_panel`.

Details

This object is included to avoid long solver runtimes during vignette building and package checking.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

`minimax_obj` *Minimize A Common Maximum Deviation From Target Goals*

Description

Implements a **minimax (goal-based)** objective that minimizes a common ceiling on deviations from multiple target values.

Let there be K objective terms with linear scores $y_k = a_k^\top x$ and target values g_k. The minimax objective minimizes the *largest deviation* from these targets across all terms.

A common deviation variable is introduced so that no objective deviates from its target by more than this value.

Usage

```
minimax_obj(x, multiple_terms, strategy_args = list())
```

Arguments

`x`	An object of class `mstATA_design` created by `mst_design()`.
`multiple_terms`	A list of objective terms created by `objective_term()`. The list must contain **two or more** objective terms. All objective terms must be **absolute (goal-based) objectives** (i.e., `goal` must be specified and non-`NULL`).
`strategy_args`	A named list of strategy-specific arguments used by the **minimax** aggregation. Supported fields: • `mode` – Character string specifying the deviation formulation. Must be either `"one_dev"` (single absolute deviation) or `"two_dev"` (separate positive and negative deviations).

Details

1. Overview

Each objective term is created using `objective_term()` and represents a linear score of the form:

$$y_k = a_k^\top x,$$

where a_k is a coefficient vector defined by the attribute, attribute level, and application scope, and x denotes the item-module-panel decision variables.

Each term has an associated target value g_k.

The minimax strategy seeks a solution that minimizes the *worst-case* deviation across all objective terms.

Key characteristics for each objective term:

(1) Categorical or Quantitative

- **Categorical**: deviation from a target number of items belonging to a specified category level;
- **Quantitative**: deviation from a target sum of quantitative item attributes (e.g., difficulty, discrimination, or item information).

(2) Applied at One of Three Levels

- **"Module-level"** – only items in a specified module contribute;
- **"Pathway-level"** – items in all modules belonging to a pathway;
- **"Panel-level"** – all item selections in the panel contribute.

(3) Only Absolute Objectives

The minimax strategy applies only to absolute (goal-based) objectives. Relative objectives without targets are not supported.

2. Deviation modes

Two equivalent deviation formulations are supported via `strategy_args$mode`:

- **One-deviation mode** (`mode = "one_dev"`):
  Introduces a single nonnegative deviation variable $d \geq 0$ representing the **largest absolute deviation** across all objectives.
  For each objective term:

$$|a_k^\top x - g_k| \leq d.$$

  The optimization problem is:

$$\min d.$$

- **Two-deviation mode** (`mode = "two_dev"`):
  Introduces two nonnegative deviation variables: $d^+ \geq 0$ (maximum overflow) and $d^- \geq 0$ (maximum shortage).
  For each objective term:

$$a_k^\top x - g_k \leq d^+$$

$$g_k - a_k^\top x \leq d^-$$

  The optimization problem is:

$$\min(d^+ + d^-).$$

Interpretation

- d (one-dev) measures the largest absolute deviation from any target;
- d^+ measures the largest amount by which any objective exceeds its target;
- d^- measures the largest amount by which any objective falls short of its target.

Value

An object of S3 class `"compiled_objective"` with named elements:

name A string indicating the objective function name.

A_binary A sparse constraint matrix (class `"Matrix"`) for binary decision variables. Rows correspond to constraints; columns correspond to binary decision variables.

A_real A sparse constraint matrix (class `"Matrix"`) for continuous decision variables. Must have the same number of rows as `A_binary`.

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary A numeric vector of objective coefficients for binary decision variables.

C_real A numeric vector of objective coefficients for continuous decision variables.

sense Always `"min"` for the minimax strategy.

decisionvar_type_new Types of auxiliary variables (continuous).

decisionvar_name_new Names of auxiliary variables.

notes Metadata including strategy name and deviation mode.

Mathematical Formulation

For $k = 1, \ldots, K$, let

$$y_k = a_k^\top x,$$

where:

- a_k is the coefficient vector defined by the objective term;
- x is the vector of binary item-module-panel decision variables;
- g_k is the target value specified by the test developer.

The minimax objective minimizes the maximum deviation across all objectives by introducing auxiliary deviation variables and solving a linear program that controls the worst-case mismatch.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool        = mini_itempool,
  design          = "1-3-3",
  exclude_pathway = c("1-1-3", "1-3-1"),
  pathway_length  = 8
)

obj1 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=-1)",
```

```
  applied_level = "Module-level",
  which_module = 1,
  sense = "min",
  goal = 12
)

obj2 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=0)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "min",
  goal = 15
)

obj3 <- objective_term(
  x = test_mstATA,
  attribute = "iif(theta=1)",
  applied_level = "Module-level",
  which_module = 1,
  sense = "min",
  goal = 12
)

# One-deviation minimax
minimax_obj(x = test_mstATA,
           multiple_terms = list(obj1,obj2,obj3),
           strategy_args = list(mode = "one_dev"))

# Two-deviation minimax
minimax_obj(x = test_mstATA,
            multiple_terms = list(obj1,obj2,obj3),
            strategy_args = list(mode = "two_dev"))
```

`mini_itempool` *Example mini item pool for MST*

Description

A simulated mini item pool for demonstrating MST functionality.

Usage

```
data(mini_itempool)
```

Format

A data frame with 30 rows and 7 columns:

item_id Unique identifier for each item

content Content area classification (e.g., `content1`, `content2`, `content3`, `content4`).

itemtype Item format (e.g., `MC`, `TEI`).

time Expected response time for the item.

discrimination Discrimination parameter (a) for the 3PL model.

difficulty Difficulty parameter (b) for the 3PL model.

guessing Guessing parameter (c) for the 3PL model.

model Item response theory (IRT) model used for the item (3PL).

nrCat Integer specifying the maximum number of response categories generated for the item.

enemy_cluing enemy information 2

enemy_similarity enemy information 1

stimulus associated stimulus information

iif(theta=-1) item information value at -1

iif(theta=0) item information value at 0

iif(theta=1) item information value at 1

Examples

```
data(mini_itempool)
```

`mixed_format_pool` *Example Mixed-Format Item Pool for MST*

Description

A simulated item pool containing both standalone items and stimulus-based items. This dataset is designed to demonstrate multistage testing (MST) functionality under a mixed-format item pool, including stimulus-based selection, multiple-objective formulations, and multiple-panel assembly.

Usage

```
data(mixed_format_pool)
```

Format

A data frame with 1000 rows and the following variables:

item_id Unique identifier for each item.

enemyitem Identifier(s) of items that cannot be administered together (enemy-item information).

content Content area classification (e.g., `content 1`, `content 2`, `content 3`, `content 4`).

dok Depth-of-Knowledge (DOK) level (`dok 1`, `dok 2`, or `dok 3`).

itemtype Item type (e.g., `MC` for multiple choice, `TEI` for technology-enhanced item).

time Estimated item response time.

discrimination Discrimination parameter (a) for the 2PL model.

difficulty Difficulty parameter (b) for the 2PL model.

model Item response theory (IRT) model for all items (`2PL`).

stim Identifier of the associated stimulus (if applicable).

pivot Indicator for pivot items. A pivot item is selected if and only if its associated stimulus is selected; non-pivot items are `NA`.

stimtype Stimulus type (e.g., `graphic-based`, `text-based`).

stimcomplexity complexity score of the associated stimulus.

Examples

```
data(mixed_format_pool)
```

`module_score_dist` *Module Score Distribution at One or More Ability Values*

Description

Compute the conditional score distributions in a module given one or more ability values (θ).

This function assumes that item category probabilities have already been computed and are provided on a common category scale.

Usage

```
module_score_dist(icc)
```

Arguments

`icc`	A named list of item-by-category probability matrices evaluated at one or more ability values. Each element corresponds to a single ability value θ and must be a numeric matrix with rows representing items and columns representing score categories (`cat0`, `cat1`, ...). For mixed dichotomous and polytomous modules, dichotomous items must be padded so that all matrices share the same set of category columns, with higher-category probabilities set to zero.

Details

Consider a module consisting of I items. Let the score for item i be a discrete random variable Y_i taking values $\{0, 1, \ldots, m_i\}$. Conditional on ability θ, assume local independence:

$$\Pr(Y_1, \ldots, Y_I \mid \theta) = \prod_{i=1}^{I} \Pr(Y_i \mid \theta).$$

The total module score is

$$S = \sum_{i=1}^{I} Y_i,$$

with support $\{0, \ldots, \sum_i m_i\}$. For a fixed ability level θ, the conditional distribution of S is obtained by discrete convolution of the item-level category probability vectors:

$$\Pr(S = s \mid \theta) = (p_1 * p_2 * \cdots * p_I)(s),$$

where $p_i = (p_{i0}(\theta), \ldots, p_{im_i}(\theta))$ and $*$ denotes convolution.

This function applies the above operation independently at each ability level.

Value

A numeric matrix giving conditional module score distributions at target theta values. Rows correspond to total scores $s = 0, \ldots, \sum_i m_i$. Columns correspond to ability values θ. Each column sums to one.

Examples

```
## Example 1: Dichotomous model
data("mini_itempool")
icc_mat<-compute_icc(mini_itempool,list("3PL"=c("discrimination","difficulty","guessing")),
            theta = 0,model_col = "model",D = 1.7)

icc_list<-compute_icc(mini_itempool,list("3PL"=c("discrimination","difficulty","guessing")),
            theta = seq(-5,5,0.1),model_col = "model",D = 1.7)

module_score_dist(icc_mat[[1]])
module_score_dist(icc_list)

## Example 2: Polytomous model (PCM)
data("poly_itempool")
icc_mat<-compute_icc(poly_itempool,list("PCM"=c("deltaj1","deltaj2","deltaj3")),
            theta = 0,model_col = "model")
icc_list<-compute_icc(poly_itempool,list("PCM"=c("deltaj1","deltaj2","deltaj3")),
            theta = seq(-5,5,0.1),model_col = "model")
module_score_dist(icc_mat[[1]])
module_score_dist(icc_list)

## Example 3: multiple IRT models (3PL + GPCM + GRM)
```

```
data("Rmst_pool")
icc_mat<-compute_icc(Rmst_pool,
                     item_par_cols = list("3PL"=c("a","b","c"),
                                          "GPCM" = c("alpha","delta1","delta2","delta3"),
                                            "GRM" = c("alpha","beta1","beta2")),
                      theta = 0,model_col = "model")
icc_list<-compute_icc(Rmst_pool,
                     item_par_cols = list("3PL"=c("a","b","c"),
                                          "GPCM" = c("alpha","delta1","delta2","delta3"),
                                            "GRM" = c("alpha","beta1","beta2")),
                      theta = 0,model_col = "model")
module_score_dist(icc_mat[[1]])
module_score_dist(icc_list)
```

`mst_design` *Create an mstATA_design Object*

Description

Constructs an `mstATA_design` object from an item pool and a multistage test (MST) design. The resulting object encodes the full structural information required for automated test assembly, including module structure, pathway structure, decision variables, and optional **preprocessing** artifacts such as stimulus mappings and enemy relationships.

Usage

```
mst_design(
  itempool,
  item_id_col = "item_id",
  design,
  rdps = NULL,
  diff_levels = NULL,
  exclude_pathways = NULL,
  module_length = NULL,
  pathway_length = NULL,
  item_module_eligibility = NULL,
  pivot_stim_map = NULL,
  enemyitem_set = NULL,
  enemystim_set = NULL
)
```

Arguments

`itempool`	A `data.frame` containing the item pool. Each row represents one item and may include metadata such as item ID, difficulty, content category, or stimulus membership.
`item_id_col`	A character string giving the column name in `itempool` that uniquely identifies items.

`design` A character string specifying the MST design. The string encodes the number of modules per stage and may use commas, dashes, or slashes as separators. For example, "1-3-3", "1,3,3", and "1/3/3" all define an MST with 1 module in stage 1, 3 modules in stage 2, and 3 modules in stage 3.

`rdps` Optional list specifying routing decision points for each stage transition in a multistage test. Default is NULL. See Details.

`diff_levels` Optional named list of difficulty labels, indexed by the number of modules in each stage. If `NULL`, default labels are used for up to five modules per stage. If any stage has more than five modules and `diff_levels` is not provided, difficulty labels are omitted.

`exclude_pathways`
An optional character vector identifying disallowed pathways. Each element must be a string of the form "x-y-z", where each number denotes the module selected at a given stage. Module indices must correspond to those implied by the MST design. If `NULL` (default), all pathways implied by the design are allowed. This option is commonly used to prohibit extreme routing transitions (e.g., from very easy to very hard modules).

`module_length` Optional numeric vector giving the number of items in each module. The length must match the number of modules implied by `design`.

`pathway_length` Optional numeric scalar specifying the total number of items in each pathway. All pathways must have equal length if this argument is used.

`item_module_eligibility`
Optional named list specifying item eligibility by module. List names correspond to (a subset of) module indices (within a single panel), and each element is a vector of item indices eligible for that module. If omitted, all items are eligible for that module.

`pivot_stim_map` Optional. A precomputed stimulus-item mapping created by create_pivot_stimulus_map(). Required for stimulus-based constraints.

`enemyitem_set` Optional. A precomputed enemy item set created by create_enemy_sets() and optionally combined via concat_enemy_sets(). Required for enemy-item constraints.

`enemystim_set` Optional. A precomputed enemy stimulus set created by create_enemy_sets() and optionally combined via concat_enemy_sets(). Required for enemy-stimulus constraints.

Details

1. Routing decision points

The list must have length `NumStages - 1`. Each element corresponds to routing after a given stage and must be a numeric vector whose length equals the number of modules in the next stage minus one.

For example, in a three-stage 1-3-3 design:

```
rdps = list(
  c(-0.5, 0.5),  # routing after stage 1
  c(-1, 1)       # routing after stage 2
)
```

The ordering of values within each vector defines adjacency among modules in the next stage.

2. module_length or pathway_length

Exactly one of `module_length` or `pathway_length` must be specified.

- If `module_length` is provided, modules at the same stage are checked (must have equal length). Pathway lengths are computed automatically.
- If `pathway_length` is provided, all pathways are assumed to have equal length and module lengths are not explicitly stored.
- If both are provided, consistency between the implied and supplied pathway lengths is checked.

3. item_module_eligibility set

By default, all items are eligible for all modules, yielding a fully flexible module-explicit formulation with decision variables $x_{i,m}$ indicating whether item i is selected in module m in a panel. Panel-level item exposure constraints ensure uniqueness across modules within a pathway/panel.

When `item_module_eligibility` is supplied, decision variables are created only for eligible item-module combinations. This reduces model size while preserving flexibility near module boundaries and avoids rigid item partitioning in original hybrid assembly strategy (Xiong, 2018).

Value

An object of class `"mstATA_design"` with the following components:

ItemPool The original item pool.

item_id_col A character string giving the column name in `ItemPool` that uniquely identifies items.

pivot_stim_map Optional verified stimulus mapping object with components `pivot_item_id`, `stimulus_name`, `stimulus_members`, and `numItems_stimulus`.

enemyitem_set Optional verified enemy item set containing `ExclusionPair`, `EnemySet`, and derived `ItemIndex`.

enemystim_set Optional verified enemy stimulus set containing `ExclusionPair`, `EnemySet`, and derived `PivotIndex`.

NumStages Number of stages in the MST design.

NumModules Number of modules in the MST design.

NumPathways Number of allowed pathways.

RDP Optional verified routing decision points.

ModuleIndex A `data.frame` describing module structure by stage. If `module_length` is supplied, includes a `ModuleLength` column.

PathwayIndex A `data.frame` describing allowed pathways and their total lengths. If difficulty labels are available, includes pathway labels.

decisionvar_name Character vector of decision variable names (one panel).

decisionvar_type Character vector of decision variable types (currently all binary).

References

Xiong, X. (2018). A hybrid strategy to construct multistage adaptive tests. *Applied Psychological Measurement*, 42(8), 630-643.doi:10.1177/0146621618762739

See Also

create_pivot_stimulus_map(),

create_enemy_sets(),

concat_enemy_sets()

Examples

```
# mini item pool
data("mini_itempool")
# Example 1: 1-3-3 design with varying module lengths
mst_design(itempool = mini_itempool,design = "1-3-3",
               module_length = c(6, 4, 4, 4, 3, 3, 3),pathway_length = NULL)

# Example 2: 1-3-3 design with fixed pathway length and excluded REH and RHE pathways
mst_design(itempool = mini_itempool,design = "1-3-3",
               exclude_pathways = c("1-1-3", "1-3-1"),
               module_length = NULL,pathway_length = 13)
# Example 3: 1-3-3 design, with item_module_eligibility provided
item_module_eligibility<-list(`1`=1:nrow(mini_itempool),
                              `2`=which(mini_itempool$difficulty<=0),
                              `4`=which(mini_itempool$difficulty>=0))
### selected items in routing module no item-level requirement,
### selected items in S2E, difficulty below 0
### selected items in S2H, difficulty above 0

mst_design(itempool = mini_itempool,design = "1-3-3",
           module_length = c(6,4,4,4,3,3,3),
           item_module_eligibility = item_module_eligibility)
```

---

mst_structure_con *Construct MST Structural Constraints for an MST Panel*

Description

Translates the structural specifications of a MST panel into a set of mandatory linear constraints that must be satisfied regardless of the assembly strategy (e.g., bottom-up, top-down, or hybrid).

The resulting constraints enforce:

- Module-level item count requirements or Pathway-level item count + module min/max + equal module length within each stage requirements
- Information balance at routing decision points (RDPs), if specified.

These constraints are intrinsic to the MST panel configuration and are independent of the assembly strategy.

Usage

```
mst_structure_con(x, stage_length_bound = NULL, info_tol = 0.5)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().
stage_length_bound	Optional bounds on the total number of items in each stage. This argument is passed to test_itemcount_con().
info_tol	A single positive numeric value specifying the allowable difference in information between adjacent modules at each routing decision point. This argument is passed to test_rdp_con().

Details

This function bundles structural constraints that are required for all MST panel assembly problems.

When routing constraints are requested, the item pool must already contain item-level information functions of the form `iif(theta = theta_point)`. These can be prepared using compute_iif().

Value

An object of S3 class "mstATA_constraint" with named elements:

name A character vector indicating the specifications in each row of A_binary

specification A data.frame summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of A_binary.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

See Also

mst_design(),

test_itemcount_con(),

test_rdp_con(),

compute_iif()

Examples

```
data("mini_itempool")
iif_matrix<-compute_iif(mini_itempool,
                item_par_cols = list("3PL"=c("discrimination","difficulty","guessing")),
                       theta = c(-0.5,0.5),model_col = "model")
mini_itempool[,paste0("iif(theta=",c(-0.5,0.5),")")]<-iif_matrix
```

```
test_mstATA <- mst_design(
  itempool = mini_itempool,module_length = c(3,4,4,4),
  design = "1-3",rdps = list(c(-0.5,0.5))
)
## build mandatory structural constraints: module-level item count and rdp constraint
mst_structure_con(x = test_mstATA,info_tol = 0.4)
```

`multipanel_spec` *Define a Multi-Panel mstATA Model*

Description

Constructs a **multi-panel ATA model** by expanding single-panel specifications to multiple panels and turning it into a solver-ready formulation.

Solution-level constraints and decision-variable linking between item-module-panel variables and solution-level variables are handled at this stage.

Usage

```
multipanel_spec(
  x,
  panel_model,
  num_panels,
  solution_con = NULL,
  global_min_use = 0,
  global_max_use = Inf,
  item_min_use = NULL,
  item_max_use = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`panel_model`	An object of class `"mstATA_model"` created by `onepanel_spec()`.
`num_panels`	Integer. Number of panels to be assembled.
`solution_con`	Optional list of `mstATA_constraint` objects defining solution-level requirements across panels (e.g., unique item/stimulus counts across panels, unique item/stimulus counts from specific categories across panels).
`global_min_use`	Scalar. Minimum number of times any item may be used across panels. Default = `0`.
`global_max_use`	Scalar. Maximum number of times any item may be used across panels. Default = `Inf`.
`item_min_use`	Optional data frame with columns `item_id` and `min`, specifying item-specific minimum reuse counts.
`item_max_use`	Optional data frame with columns `item_id` and `max`, specifying item-specific maximum reuse counts.

Details

1. Parallel Panels

The single-panel specification is expanded to `num_panels` panels by replicating constraint matrices in a block-diagonal structure. All panels share identical constraints and objectives.

Solution-level constraints, if provided, are appended after panel expansion.

Specification traceability

The returned model contains an expanded specification table that records, for each requirement:

- the originating panel,
- the number of constraints generated,
- the contiguous row ranges in the final solver matrix.

This information can be used to diagnose infeasibility and to prioritize or relax groups of constraints programmatically.

2. Consistency between item-module-panel selection and solution-level item indicators

The decision-variable linking constraints (e.g., dvlink_item_solution()) are automatically added to link item-module-panel selection and solution-level item selection.

The constraint enforces:

Each item can be used at least a specified minimum number of times and/or no more than a specified maximum number of times across all panels.

Key characteristics:

- Attribute: *itemset level (item-itself set) logical* (if selected, then increment reuse).
- Constraints are applied at the **"Solution-level"**.

Global reuse limits are supplied via `global_min_use` and `global_max_use`. These bounds apply to all items unless overridden.

Item-specific reuse limits may be supplied through:

- `item_min_use`: a data frame with columns `item_id` and `min`
- `item_max_use`: a data frame with columns `item_id` and `max`

The total number of constraints generated is always $2 \times P$ where P is the pool size, corresponding to one minimum and one maximum constraint per item.

Value

An object of class `"mstATA_model"`, represented as a named list containing constraint matrices, objective coefficients, decision variable metadata, bounds, and optimization sense.

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse constraint matrix (class `"Matrix"`) for binary decision variables. Rows correspond to constraints; columns correspond to binary decision variables.

A_real A sparse constraint matrix (class `"Matrix"`) for continuous decision variables. Must have the same number of rows as `A_binary`.

A Constraint coefficient matrix (sparse or dense).

C_binary A numeric vector of objective coefficients for binary decision variables.

C_real A numeric vector of objective coefficients for continuous decision variables.

obj Objective function coefficient vector.

d Right-hand-side vector of constraints.

operators Constraint operators (e.g., `"<="`, `">="`, `"="`).

vtype Decision variable types (`"B"` for binary, `"C"` for continuous).

varname Decision variable names.

lb Lower bounds for decision variables.

ub Upper bounds for decision variables.

sense Optimization sense (`"min"` or `"max"`).

See Also

`onepanel_spec()`,
`dvlink_item_solution()`

`objective_term` *Create a Single Linear Objective Term*

Description

Creates a linear objective term of the form $a^\top x$, where the vector a represents categorical or quantitative attribute values for the involved items (or stimuli), and the decision variables x represent item selections within a single MST panel.

Depending on the attribute supplied, the objective can target:

- maximize or minimize the number of items belong to a specific category level, or
- maximize or minimize the sum of quantitative attribute values across items (e.g., difficulty, IIF values, discrimination).

Objective terms may be applied at three hierarchical levels:

- **"Module-level"** – aggregates only items belonging to one specific module;
- **"Pathway-level"** – aggregates items in all modules that compose a single MST pathway; and
- **"Panel-level"** – aggregates all item selections across all modules in a panel.

The function supports two types of objectives:

- **Relative objective** - maximize or minimize $a^\top x$.
- **Absolute deviation objective** - minimize $|a^\top x - g|$ where g is the target value.

Usage

```
objective_term(
  x,
  attribute,
  cat_level = NULL,
  applied_level = c("Module-level", "Pathway-level", "Panel-level"),
  which_module = NULL,
  which_pathway = NULL,
  sense = "max",
  goal = NULL
)
```

Arguments

x	An object of class "mstATA_design" created by "mst_design()".
attribute	A string giving the column name in x$ItemPool that contains the attribute used in the objective term. This may be categorical or quantitative.
cat_level	A character string or NULL. If a character string is supplied, the attribute is treated as categorical and the objective counts items belonging to this category. If NULL, the attribute is treated as quantitative and the objective uses the numeric attribute values.
applied_level	One of: "Module-level", "Pathway-level", "Panel-level". See *Details* for the rules governing module/pathway selection.
which_module	Integer scalar specifying the module to which the objective applies (only permitted when applied_level = "Module-level").
which_pathway	Integer scalar specifying the pathway to which the objective applies (only permitted when applied_level = "Pathway-level").
sense	One of "max" or "min". For absolute deviation objectives, "min" is enforced automatically.
goal	Optional numeric scalar. If supplied, the objective minimizes the absolute deviation from the target: $\|a^\top x - g\|$ where g is the target value. If NULL, a standard maximize/minimize objective is created.

Details

1. Application scope

- **Module-level:** which_module must be supplied and must be a scalar. which_pathway must be NULL.
- **Pathway-level:** which_pathway must be supplied and must be a scalar. which_module must be NULL.
- **Panel-level:** Neither which_module nor which_pathway may be supplied.

2. Objective meaning

If cat_level = NULL, the vector a consists of quantitative attribute values. If cat_level is specified, a is an indicator vector for items in the corresponding category.

The objective term therefore represents:

- the **sum of attribute values** (quantitative), or
- the **count of items** in a category (categorical).

Value

An object of class `"mstATA_objective"` with components:

name Character description of the objective term.

coef_val A 1-row sparse matrix containing the objective coefficients.

goal The target value for absolute deviation objectives, or `NULL`.

sense The optimization direction (`"max"` or `"min"`).

Examples

```
data("mini_itempool")
test_mstATA <- mst_design(
  itempool       = mini_itempool,
  design         = "1-3-3",
  exclude_pathway = c("1-1-3","1-3-1"),
  pathway_length = 8
)

# Example 1: Maximize information at ability = 0 in the routing module
objective_term(
  x = test_mstATA,
  attribute = "iif(theta=0)",
  cat_level = NULL,
  applied_level = "Module-level",
  which_module = 1,
  sense = "max"
)

# Example 2: Target a test information value at ability = -1 for pathway 1
objective_term(
  x = test_mstATA,
  attribute = "iif(theta=-1)",
  cat_level = NULL,
  applied_level = "Pathway-level",
  which_pathway = 1,
  sense = "max",
  goal = 10
)

# Example 3: Maximize the number of MC items in a panel
objective_term(
  x = test_mstATA,
  attribute = "itemtype",
  cat_level = "MC",
  applied_level = "Panel-level",
  sense = "max"
)
```

onepanel_spec *Define a Single-Panel mstATA Model*

Description

Constructs a **single-panel ATA specification** by assembling constraints, optional objective formulations, and required decision-variable linking rules into a solver-ready model with full semantic traceability.

This function validates that all supplied constraints apply to a *single* panel, automatically injects decision-variable linking constraints when required, and records row-index ranges that map semantic requirements to solver constraint rows.

Usage

```
onepanel_spec(x, constraints, objective = NULL)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().
constraints	A list of "mstATA_constraint" objects defining requirements for assembling **one panel**. Constraints that explicitly reference multiple panels (i.e., solution-level constraints across panels) are not allowed and will be rejected.
objective	Optional compiled objective of class "compiled_objective". Objectives may introduce auxiliary decision variables and additional constraints, and are only permitted for operational (single-panel) specifications.

Details

1. Constraint scope

All constraints supplied to onepanel_spec() must apply to a single panel. This includes requirements defined at the item, stimulus, itemset, module, pathway, or panel level.

Constraints that operate *across panels* (solution-level reuse or exposure constraints) must be specified during multi-panel expansion and are not permitted in a single-panel specification.

2. Automatic decision-variable linking

If module-level logical constraints involving item-stimulus relationships are detected, and stim_itemcount_con() is not included, onepanel_spec() automatically injects item-stimulus gating constraints via all-in/all-out selection. This ensures that item-level and stimulus-level selections remain logically consistent.

3. Row-index traceability

All constraint rows–including those introduced by objectives and automatically injected decision-variable linking constraints–are assigned contiguous row-index ranges in the compiled constraint matrix.

These ranges are recorded in the returned specification table, allowing users to:

- diagnose infeasibility,
- identify binding constraint groups, and
- programmatically relax or prioritize subsets of constraints.

Value

An object of class `"mstATA_model"`, represented as a named list containing constraint matrices, objective coefficients, decision variable metadata, bounds, and optimization sense.

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse constraint matrix (class `"Matrix"`) for binary decision variables. Rows correspond to constraints; columns correspond to binary decision variables.

A_real A sparse constraint matrix (class `"Matrix"`) for continuous decision variables. Must have the same number of rows as `A_binary`.

A Constraint coefficient matrix (sparse or dense).

C_binary A numeric vector of objective coefficients for binary decision variables.

C_real A numeric vector of objective coefficients for continuous decision variables.

obj Objective function coefficient vector.

d Right-hand-side vector of constraints.

operators Constraint operators (e.g., `"<="`, `">="`, `"="`).

vtype Decision variable types (`"B"` for binary, `"C"` for continuous).

varname Decision variable names.

lb Lower bounds for decision variables.

ub Upper bounds for decision variables.

sense Optimization sense (`"min"` or `"max"`).

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)

test_mstATA <- mst_design(
  itempool = reading_itempool,
  design   = "1-3-3",
  module_length = c(14,12,12,12,12,12,12),
  pivot_stim_map = pivot_stim_map
)
spec1<-test_itemcount_con(x=test_mstATA)
```

```
spec2<-test_stimcat_con(
  x = test_mstATA,
  attribute = "stimulus_type",
  cat_levels = c("history","social studies"),
  operator = "=",
  target_num = c(1,1),
  which_module = 1)
spec3<-test_stimcat_con(
  x = test_mstATA,
  attribute = "stimulus_type",
  cat_levels = "history",
  operator = "=",
  target_num = 2,
  which_module = 2:4)
spec4<-test_stimcat_con(
  x = test_mstATA,
  attribute = "stimulus_type",
  cat_levels = "social studies",
  operator = "=",
  target_num = 2,
  which_module = 5:7)
spec5<-stim_itemcount_con(
  x = test_mstATA,
  min = 5,
  max = 7
)
obj_term<-objective_term(x = test_mstATA,attribute = "discrimination",
                         applied_level = "Module-level",which_module = 1,sense = "max")

panel_spec <- onepanel_spec(
  x = test_mstATA,
  constraints = list(spec1,spec2,spec3,spec4,spec5),
  objective = single_obj(x=test_mstATA,single_term = obj_term)
)
```

`panel_itemcat_con` *Generate Panel-Level Constraints for the Min/Exact/Max Number of Items from Specific Categorical Levels*

Description

This function generates a linear constraint matrix that restricts the number of items belonging to specified categorical levels (e.g., content area, format type) across all modules/pathways in an MST panel.

The constraint may enforce:

- a minimum number of items in category c,
- a maximum number of items in category c, or

- an exact (equality) requirement for category c.

The total number of constraints equals the number of category levels supplied in `cat_levels`.

Usage

```
panel_itemcat_con(x, attribute, cat_levels, operator, target_num)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that contains the **item categorical attribute**.
`cat_levels`	A character string or vector of categorical attribute levels. These define the item categories being constrained.
`operator`	A character string indicating the type of constraint. Must be one of `"<="`, `"="`, or `">="`.
`target_num`	A scalar or vector specifying the required number of items for each category in `cat_levels`.

Details

1. Specification

The constraint enforces:

A minimum, exact, or maximum number of items from specified categorical levels must be selected within a panel.

Key characteristics:

- The attribute type is *categorical*.
- The attribute is defined at the *item level* in the item pool.
- The constraints are applied at the *Panel-level*.

Users may restrict the constraints to any subset of categories via the `cat_levels` argument.

Value

An object of S3 class `"constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

C_binary NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_c^{item} denote the set of items that belong to category c.

$$\sum_{m} \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} \overset{\leq}{\geqq} n_c^{item}, s = 1, \ldots, S, i_s = 1, \ldots, I_s.$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- The stacked operator, $\overset{\leq}{\geqq}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- n_c^{item} specify the required number of items from category level c in a panel.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool      = mini_itempool,
  design        = "1-3-3",
  module_length = c(4,2,2,2,2,2,2)
)

# Example 1: Require at least 10 multiple-choice items in the panel
panel_itemcat_con(
  x         = test_mstATA,
  attribute = "itemtype",
  cat_levels    = "MC",
  operator  = ">=",
  target_num = 10
)

# Example 2: Require at least 10 multiple-choice items, at least 3 TEI items in the panel
panel_itemcat_con(
  x         = test_mstATA,
  attribute = "itemtype",
  cat_levels    = c("MC","TEI"),
  operator  = ">=",
```

```
  target_num = c(10,5)
)
```

`panel_itemreuse_con` *Generate Panel-Level Constraints for Item Reuse Across Modules/Pathways*

Description

This function constructs panel-level constraints that limit how many times an item may be selected across the modules and pathways contained within a single MST panel.

Two reuse-control modes are supported via `overlap`:

- **overlap = FALSE** – No item reuse anywhere in the panel (the item may appear in at most one module across the entire panel).
- **overlap = TRUE** – Item reuse is permitted across adaptive modules within the same stage, but items must remain unique within each individual pathway.

If `item_module_eligibility = NULL` in `mst_design()`, the total number of constraints depends on `overlap`:

- **overlap = FALSE:**
  The number of constraints = number of items in the item pool.
- **overlap = TRUE:**
  The number of constraints = (number of items) x (number of pathways).

Usage

```
panel_itemreuse_con(x, overlap = FALSE)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`overlap`	Logical. If `FALSE` (default), an item may appear in at most one module in a panel. If `TRUE`, item reuse within the same stage is allowed, but items must be unique within each pathway.

Details

1. Specification

The constraint enforces:

An item can be selected at most once within a pathway/panel.

Key characteristics:

- The attribute type is logical (if..., then...) with *item identifier*: each item has an unique ID.
- The attribute is defined at the *itemset level* (item-itself set) in the item pool.

- The constraint is enforced at the **Panel-level**, i.e., across all modules and pathways in a panel.

2. Interpretation of overlap

(a) `overlap = FALSE` **– Strict item uniqueness within the entire panel:**
Under this condition:

- An item may appear in at most one module anywhere in the panel.
- Selecting the item in any module automatically prevents its selection in all other modules of the panel.
- The overlap among pathways are only due to the MST structure.

(b) `overlap = TRUE` **– Item reuse allowed within stage, but not within pathway:**
Under this condition:

- Items **may** be reused across adaptive modules in the same stage.
- Items **must not** be reused within the same pathway (i.e., an item may appear at most once per pathway).
- The overlap among pathways not only due to the MST structure, but also due to some common items across modules in the same stage.

Example interpretation for a 1-3-3 design::

- (1) If an item is selected in the routing module implies it cannot appear in any S2 or S3 module.
- (2) If selected in an S2 module implies it may appear in other S2 modules (same stage), but not in S1R or any S3 module.
- (3) If selected in an S3 module implies it may appear in other S3 modules, but not in S1R or any S2 module.

3. Why This Is a Panel-Level Constraint
Although the constraint logic differs for the two overlap conditions, both versions govern item allocation across the set of modules that comprise a full panel. In other words, the constraint concerns **how an item may or may not appear within the assembled panel as a whole**, rather than applying to a specific module or pathway in isolation. For this reason, both cases are formally categorized as *Panel-level* item-reuse constraints.

Value

An object of S3 class `"constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Case 1: `overlap` = `TRUE` Item must be unique within each pathway:

$$\sum_{m \in r} x_{i_s,m} \leq 1$$

Case 2: `overlap` = `FALSE` Item must be unique across the entire panel:

$$\sum_{m} x_{i_s,m} \leq 1$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool      = mini_itempool,
  design        = "1-3-3",
  module_length = c(4,2,2,2,2,2,2)
)

# Example 1: No item overlap within a panel
panel_itemreuse_con(x = test_mstATA, overlap = FALSE)

# Example 2: Allow item reuse within the same stage
panel_itemreuse_con(x = test_mstATA, overlap = TRUE)


# Example 3: item_module_eligibility supplied -- disjoint module eligibility
# No need for panel_itemreuse_con() in this case
item_module_eligibility <- list(
  `1` = intersect(which(mini_itempool$difficulty >= -1),
                  which(mini_itempool$difficulty < 0)),
  `2` = intersect(which(mini_itempool$difficulty >= -2),
                  which(mini_itempool$difficulty < -1)),
  `3` = intersect(which(mini_itempool$difficulty >= 0),
                  which(mini_itempool$difficulty < 1)),
  `4` = intersect(which(mini_itempool$difficulty >= 1),
                  which(mini_itempool$difficulty < 2))
)
```

```
new_test_mstATA <- mst_design(
  itempool               = mini_itempool,
  design                 = "1-3",
  module_length          = c(6,7,7,7),
  item_module_eligibility = item_module_eligibility
)

## The generated constraint matrix are redundant and will be omitted in the ATA model.
try(panel_itemreuse_con(x = new_test_mstATA, overlap = FALSE))


# Example 4: Overlapping module eligibility -- reuse constraints required
item_module_eligibility <- list(
  `1` = which(mini_itempool$difficulty < 2),
  `2` = which(mini_itempool$difficulty < 0),
  `3` = intersect(which(mini_itempool$difficulty >= -1),
                  which(mini_itempool$difficulty < 1)),
  `4` = intersect(which(mini_itempool$difficulty >= 0),
                  which(mini_itempool$difficulty < 2))
)

new_test_mstATA <- mst_design(
  itempool               = mini_itempool,
  design                 = "1-3",
  module_length          = c(6,7,7,7),
  item_module_eligibility = item_module_eligibility
)

# Items eligible for multiple modules require reuse control
panel_itemreuse_con(x = new_test_mstATA, overlap = FALSE)

# Overlap within stage implies item may appear in more modules
panel_itemreuse_con(x = new_test_mstATA, overlap = TRUE)
```

`panel_stimcat_con` *Generate Panel-Level Constraints for the Min/Exact/Max Number of Stimuli from Specific Categorical Levels*

Description

This function generates a linear constraint matrix based on a stimulus-level categorical attribute (e.g., stimulus type, passage theme) across all modules and pathways within a **single panel**.

The constraint may enforce:

- a minimum number of stimuli in category c,
- a maximum number of stimuli in category c, or

- an exact (equality) requirement in category c.

The number of linear constraints generated equals the number of category levels supplied in `levels`.

Usage

```
panel_stimcat_con(x, attribute, cat_levels, operator, target_num)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` containing the **stimulus-level categorical attribute**.
`cat_levels`	A character vector of category levels to be constrained. Must match the unique values in the specified `attribute`.
`operator`	A character string specifying the constraint direction: one of `"<="`, `"="`, or `">="`.
`target_num`	A scalar or vector giving the required number of stimuli in each category listed in `cat_levels`.

Details

1. Specification

The constraint enforces:

A minimum, exact, or maximum number of stimuli from specified categories must be selected somewhere within a panel.

Key characteristics:

- The attribute type is *categorical*.
- The attribute is defined at *stimulus-level* in the item pool.
- The constraints are applied at the *Panel-level*.

Users may restrict the constraint to specific categories using the `cat_levels` argument.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_c^{stim} denote the set of stimuli that belong to category c.

$$\sum_m \sum_{s \in V_c^{stim}} x_{i_s^*, m} \overset{\leq}{\underset{=}{\geq}} n_c^{stim}$$

Here:

- $x_{i_s^*, m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus s is selected in that module.
- The stacked operator, $\overset{\leq}{\underset{=}{\geq}}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- n_c^{stim} specify the required number of stimuli from category level c across modules/pathways in a panel.

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)

# Example: Require at most 4 history passages in an assembled panel
test_mstATA <- mst_design(
  itempool       = reading_itempool,
  design         = "1-3-3",
  module_length  = c(10,12,12,12,15,15,15),
  exclude_pathway = c("1-1-3", "1-3-1"),
  pivot_stim_map = pivot_stim_map
)

panel_stimcat_con(
  x             = test_mstATA,
  attribute     = "stimulus_type",
  cat_levels        = "history",
  operator      = "<=",
  target_num    = 4
)
```

`Pi_internal` *Compute Item Response Category Probabilities and Derivatives*

Description

Computes item response category probabilities under various IRT models at a given ability value θ.

Usage

```
Pi_internal(theta, itempar_mat, model = NULL, D = 1)
```

Arguments

`theta`	A numeric scalar specifying the ability value at which probabilities are evaluated.
`itempar_mat`	A numeric matrix with one row per item and columns corresponding to the required parameters for the specified IRT model. The required structure depends on `model`. See Details.
`model`	A character string specifying the IRT model. Supported models are: `"GRM"`, `"MGRM"`, `"PCM"`, `"GPCM"`, `"RSM"`, and `"NRM"`. If `NULL`, a dichotomous logistic model (1PL/2PL/3PL/4PL) is assumed.
`D`	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.

Details

(1) Dichotomous IRT models

For the models `"1PL"`, `"RASCH"`, `"2PL"`, `"3PL"`, and `"4PL"`, the argument `itempar_mat` must be a numeric matrix with one row per item and four columns: discrimination (a_j), difficulty (b_j), guessing (c_j),upper (d_j).

(2) Polytomous IRT models

For a polytomous item j, let g_j denote the maximum score, so that the possible response categories are $0, 1, \ldots, g_j$.

The number of response categories may vary across items under the `"GRM"`, `"PCM"`, `"GPCM"`, and `"NRM"` models. In contrast, under the `"MGRM"` and `"RSM"` models, the number of response categories is assumed to be the same for all items.

The argument `itempar_mat` must be a numeric matrix with one row per item and a number of columns determined by the specified IRT model. The required structure is:

- **GRM**: A matrix with $\max_j g_j + 1$ columns, containing parameters $(\alpha_j, \beta_{j1}, \ldots, \beta_{jg_j})$, where α_j is the discrimination parameter and β_{jk} are the ordered threshold parameters.
- **MGRM**: A matrix with $g_j + 2$ columns, containing parameters $(\alpha_j, b_j, c_{j1}, \ldots, c_{jg_j})$, where α_j is the discrimination parameter, b_j is the location parameter, and c_{jk} are category step parameters.

- **PCM**: A matrix with $\max_j g_j$ columns, containing step parameters $(\delta_{j1}, \ldots, \delta_{jg_j})$.
- **GPCM**: A matrix with $\max_j g_j + 1$ columns, containing parameters $(\alpha_j, \delta_{j1}, \ldots, \delta_{jg_j})$, where α_j is the discrimination parameter and δ_{jk} are step parameters.
- **RSM**: A matrix with $g + 1$ columns, containing parameters $(\lambda_j, \delta_1, \ldots, \delta_g)$, where λ_j is the item location parameter and δ_k are common step parameters across items.
- **NRM**: A matrix with $2 \max_j g_j$ columns, containing parameters $(\alpha_{j1}, c_{j1}, \alpha_{j2}, c_{j2}, \ldots, \alpha_{jg_j}, c_{jg_j})$, where α_{jk} and c_{jk} are the slope and intercept parameters for category k.

Value

A list with components:

Pi A matrix of category response probabilities. Rows correspond to items and columns correspond to categories.

dPi Matrix of first derivatives ∂P_{jk}.

`plot_panel_tcc` *Plot Test Category Probability Curves*

Description

Plots category probability curves aggregated at the module or pathway level.

The function supports both single-panel and multi-panel visualization and automatically adjusts color and faceting to facilitate meaningful comparisons across modules, pathways, and panels.

Usage

```
plot_panel_tcc(
  assembled_panel,
  item_par_cols,
  model_col,
  D = 1,
  theta = seq(-5, 5, 0.1),
  unit = "module",
  mode = "within_panel",
  legend = TRUE
)
```

Arguments

`assembled_panel`
: A `"mstATA_panel"` object returned by assembled_panel(). All panels must share identical module and pathway structure.

item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in items[[model_col]]: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names that contain the required item parameters for the corresponding model.
model_col	A character string specifying the column name in either ItemsInModules or ItemsInPathways data frames that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", or "NRM".
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.
theta	A numeric vector of ability values. Default is seq(-5,5,0,1).
unit	The aggregation unit to display. Must be either "module" or "pathway".
mode	Plotting mode for multi-panel results. Must be either "within_panel" or "across_panel". Default is "within_panel".
legend	Logical; whether to display the legend (default is TRUE).

Details

For a single panel, category probability curves are colored by module or pathway.

For multiple panels, two visualization modes are supported:

- "within_panel": curves are colored by module or pathway and faceted by panel.
- "across_panel": curves are colored by panel and faceted by module or pathway.

For dichotomously-scored items (two response categories), only the probability curve for the correct-response category (cat1) is displayed.

For polytomously-scored items (more than two response categories), probability curves for all response categories are displayed.

Category probability curves represent the average probability of each response category across items within the selected module or pathway at each ability value.

Value

A ggplot object.

plot_panel_tif *Plot Test Information Functions*

Description

Plots test information functions (TIFs) aggregated at the module or pathway level.

The function supports both single-panel and multi-panel visualization and automatically adjusts color and faceting to facilitate meaningful comparisons across modules, pathways, and panels.

Usage

```
plot_panel_tif(
  assembled_panel,
  item_par_cols,
  model_col,
  D = 1,
  theta = seq(-5, 5, 0.1),
  unit = "module",
  mode = "within_panel",
  legend = TRUE
)
```

Arguments

assembled_panel	A "mstATA_panel" object returned by assembled_panel(). All panels must share identical module and pathway structure.
item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in items[[model_col]]: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names that contain the required item parameters for the corresponding model.
model_col	A character string specifying the column name in either ItemsInModules or ItemsInPathways data frames that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", or "NRM".
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.
theta	A numeric vector of ability values.
unit	The aggregation unit to display. Must be either "module" or "pathway". Default is "module".
mode	Plotting mode for multi-panel results. Must be either "within_panel" or "across_panel". Default is "within_panel".
legend	Logical; whether to display the legend (default is TRUE).

Details

For a single panel, information curves are colored by module or pathway.

For multiple panels, two visualization modes are supported:

- `"within_panel"`: curves are colored by module or pathway and faceted by panel.
- `"across_panel"`: curves are colored by panel and faceted by module or pathway.

Value

A `ggplot` object.

---

plot_tif *Plot Test Information Functions*

Description

Plots testing information functions (TIFs).

Usage

```
plot_tif(items, item_par_cols, theta = seq(-3, 3, 0.1), model_col, D = 1)
```

Arguments

`items`	A data frame containing item metadata. Each row represents an item. The data frame must include a column specifying the item model (see `model_col`) and the parameter columns referenced in `item_par_cols`.
`item_par_cols`	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in `items[[model_col]]`: `"1PL"`, `"RASCH"`, `"2PL"`, `"3PL"`, `"4PL"`, `"GRM"`, `"MGRM"`, `"PCM"`, `"GPCM"`, `"RSM"`, and `"NRM"`. Each list element is a character vector giving the column names in `items` that contain the required item parameters for the corresponding model.
`theta`	A numeric vector of ability values at which the IIFs are evaluated.
`model_col`	A character string specifying the column name in `items` that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: `"1PL"`, `"RASCH"`, `"2PL"`, `"3PL"`, `"4PL"`, `"GRM"`, `"MGRM"`, `"PCM"`, `"GPCM"`, `"RSM"`, or `"NRM"`.
`D`	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.

Value

A `ggplot` object.

poly_itempool *Example Polytomous Item Pool for MST*

Description

A simulated item pool consisting exclusively of polytomously scored items, designed to demonstrate multistage testing (MST) functionality under a partial credit modeling framework.

Usage

```
data(poly_itempool)
```

Format

A data frame with 50 rows and the following columns:

item_id Unique identifier for each item.

content Content area classification (e.g., `content1`, `content2`, `content3`, `content4`).

deltaj1 First step (threshold) parameter in the Partial Credit Model (PCM).

deltaj2 Second step (threshold) parameter in the Partial Credit Model (PCM).

deltaj3 Third step (threshold) parameter in the Partial Credit Model (PCM).

model Item response theory (IRT) model used for all items (`PCM`).

nrCat Integer specifying the maximum number of response categories generated for the item.

Examples

```
data(poly_itempool)
```

print.mstATA_model *Print mstATA Model*

Description

Prints a concise, human-readable summary of a compiled `mstATA_model` object which is produced by onepanel_spec() or multipanel_spec().

This print method is a **diagnostic and inspection utility**. It reports:

- the number of binary and continuous decision variables,
- the total number of linear constraints,
- the objective sense (`"min"` or `"max"`),
- and a formatted specification table mapping semantic requirements to solver constraint rows.

The specification table provides row-range traceability for debugging infeasibility and constraint prioritization.

Usage

```
## S3 method for class 'mstATA_model'
print(x, ...)
```

Arguments

x	An object of class "mstATA_model".
...	Further arguments passed to or from other methods.

Value

The input object x, returned invisibly.

See Also

onepanel_spec(),
multipanel_spec()

---

reading_itempool *Reading Item Pool*

Description

A reading item pool for demonstrating MST functionality.

Usage

```
data(reading_itempool)
```

Format

A data frame with 500 rows and 15 columns:

item_id Unique identifier for each item.

content Content area classification (e.g., content1, content2, content3, content4).

itemtype Item format (e.g., MC, TEI).

time Expected response time for the item.

discrimination Discrimination parameter (a) for the 3PL model.

difficulty Difficulty parameter (b) for the 3PL model.

guessing Guessing parameter (c) for the 3PL model.

model Item response theory (IRT) model used for the item (3PL).

nrCat Integer specifying the maximum number of response categories generated for the item.

enemy_item enemy item pairs/sets information

stimulus associated stimulus information

pivot_item the indicator of pivot items. A pivot item is defined as an item that is always selected for the test if and only if its stimulus is selected

enemy_stimulus enemy stimulus pairs/sets information

stimulus_type Stimulus types: history, social studies

stimulus_words Stimulus words

Examples

```
data(reading_itempool)
```

`reading_panel` *Precomputed Stimulus-Based Assessment (MST Assembly Example)*

Description

A precomputed MST panel assembled with the HiGHS solver demonstrated in *Stimulus Based Assessment* vignettes.

Usage

```
reading_panel
```

Format

An object of class `mstATA_panel`.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

`report_test_itemcat` *Summarize Categorical Item Attributes in Assembled MST Panels*

Description

Summarizes the number of selected items belonging to specified categorical attribute levels (e.g., content area, item type) within assembled MST panels. Summaries can be produced at the module level or pathway level, depending on the constraint scope.

Usage

```
report_test_itemcat(
  assembled_panel,
  attribute,
  cat_levels,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

assembled_panel	A "mstATA_panel" object returned by assembled_panel. All panels must share identical module and pathway structure.
attribute	A character string specifying the name of a categorical attribute in the item pool (e.g., "content", "itemtype").
cat_levels	A character vector specifying the attribute levels to be summarized (e.g., "MC" or c("MC", "TEI")).
which_module	Optional integer vector specifying module indices for module-level summaries.
which_pathway	Optional integer vector specifying pathway indices for pathway-level summaries.

Details

This function operates on the output of assembled_panel() and is primarily intended for evaluating categorical test constraints.

The scope of the summary is determined by which_module and which_pathway:

- If which_module is provided, a module-level summary is returned for the specified modules.
- If which_pathway is provided, a pathway-level summary is returned for the specified pathways.
- If both which_module and which_pathway are NULL, the function defaults to a **module-level summary over all modules**.

The attribute specified by attribute must be a categorical attribute. This is verified internally using check_attribute_column(), and an error is thrown if the attribute is not categorical.

Counts are reported *separately for each requested attribute level*. For example, if cat_levels = c("MC", "TEI"), the output will contain separate rows for MC and TEI items within each module or pathway.

Value

A data frame with one row per panel-module (or panel-pathway) combination. The returned data frame contains:

panel_id Panel identifier.

module_id or pathway_id Module or pathway identifier.

cat_level_1, cat_level_2, ... Number of selected items belonging to each specified categorical level within the corresponding module or pathway. Each category level appears as a separate column.

See Also

`assembled_panel()`

`report_test_itemquant` *Summarize Quantitative Item Attributes in Assembled MST Panels*

Description

Summarizes quantitative item attributes (e.g., test information, time, difficulty) within assembled multistage test (MST) panels. The summary can be computed as either the sum or the average of the attribute values, at the module level or pathway level, depending on the constraint scope.

Usage

```
report_test_itemquant(
  assembled_panel,
  attribute,
  statistic = c("average", "sum"),
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`assembled_panel`	A `"mstATA_panel"` object returned by `assembled_panel`. All panels must share identical module and pathway structure.
`attribute`	A character string specifying the name of a **quantitative** attribute in the item pool (e.g., `"time"`, `"difficulty"`, `"information"`).
`statistic`	A character string specifying the summary statistic to compute. One of `"average"` or `"sum"`.
`which_module`	Optional integer vector specifying module indices for module-level summaries.
`which_pathway`	Optional integer vector specifying pathway indices for pathway-level summaries.

Details

This function operates on the output of `assembled_panel()` and is primarily intended for evaluating quantitative test constraints.

The scope of the summary is determined by `which_module` and `which_pathway`:

- If `which_module` is provided, a module-level summary is returned for the specified modules.

- If `which_pathway` is provided, a pathway-level summary is returned for the specified pathways.
- If both `which_module` and `which_pathway` are `NULL`, the function defaults to a **module-level summary over all modules**.

The attribute specified by `attribute` must be a quantitative attribute. This is verified internally using `check_attribute_column()`, and an error is thrown if the attribute is not quantitative.

Value

A data frame with one row per panel-module (or panel-pathway) combination. The returned data frame contains:

panel_id Panel identifier.

module_id or pathway_id Module or pathway identifier.

attribute The name of the quantitative attribute being summarized.

sum or average The computed summary statistic (sum or average) of the specified quantitative attribute within the corresponding module or pathway.

See Also

`assembled_panel()`

`report_test_tcc` *Report Test Characteristic Curves (TCC)*

Description

Computes and reports TCC curves aggregated at the module or pathway level for assembled MST panels.

Item-level category probabilities are obtained from compute_icc().

Usage

```
report_test_tcc(
  assembled_panel,
  theta,
  item_par_cols,
  model_col,
  D = 1,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

assembled_panel	A "mstATA_panel" object returned by assembled_panel. All panels must share identical module and pathway structure.
theta	A numeric vector of ability values.
item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in items[[model_col]]: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names that contain the required item parameters for the corresponding model.
model_col	A character string specifying the column name in either ItemsInModules or ItemsInPathways data frames that indicates the IRT model used for each item. Values in this column must correspond to the supported model names: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", or "NRM".
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.
which_module	Optional vector specifying which modules to include.
which_pathway	Optional vector specifying which pathways to include.

Value

A data frame containing the following columns:

panel_id Identifier of the assembled panel.

module_id / pathway_id Identifier of the module or pathway.

theta Ability level at which the statistics are evaluated.

TCC Test characteristic curve value, defined as the expected total score $\sum E[P(Y = k \mid \theta)]$.

max_score Maximum possible score for the module or pathway.

TCC_norm Normalized TCC value defined as TCC / max_score.

See Also

compute_icc()

---

report_test_tif *Report Test Information Functions (TIF)*

Description

Computes and reports test information functions aggregated at the module or pathway level for assembled MST panels.

Item information functions are computed using compute_iif(), and then summed across items within specified module or pathway.

Usage

```
report_test_tif(
  assembled_panel,
  theta,
  item_par_cols,
  model_col,
  D = 1,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

assembled_panel	A "mstATA_panel" object returned by assembled_panel. All panels must share identical module and pathway structure.
theta	A numeric vector of ability values at which information functions are evaluated.
item_par_cols	A named list defining the IRT parameter columns required for each supported model. The list names must exactly match the model identifiers specified in items[[model_col]]: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", and "NRM". Each list element is a character vector giving the column names that contain the required item parameters for the corresponding model.
model_col	A character string specifying the column name in either ItemsInModules or ItemsInPathways data frames that indicates the IRT model used for each item. Values in this column must correspond to one of the supported model names: "1PL", "RASCH", "2PL", "3PL", "4PL", "GRM", "MGRM", "PCM", "GPCM", "RSM", or "NRM".
D	Scaling constant used in the IRT model. Default is D=1 (for logistic metric); D=1.702 yields approximately the normal metric.
which_module	Optional integer vector specifying which modules to include.
which_pathway	Optional integer vector specifying which pathways to include.

Value

A data frame with one row per panel-module (or panel-pathway)-ability combination. The returned data frame contains:

panel_id Panel identifier.

module_id or pathway_id Module or pathway identifier.

theta Ability value at which test information is evaluated.

information Sum of item information functions for all selected items within the corresponding module or pathway, evaluated at the given ability value.

See Also

compute_iif()

`Rmst_pool` *Item pool from Rmst for examples*

Description

A data frame containing 400 3PL items, 20 GPCM items, and 20 GRM items. This item pool is adapted from the example item pool used to demonstrate multistage test assembly in the **Rmst** package.

Usage

```
data(Rmst_pool)
```

Format

A data frame with 440 rows and 14 variables:

Item_id Unique identifier for each item.

model IRT model specification for the item (e.g. `"3PL"`, `"GPCM"`, `"GRM"`).

a Discrimination parameter (a) for the 3PL model.

b Difficulty parameter (b) for the 3PL model.

c Guessing parameter (c) for the 3PL model.

alpha Discrimination parameter (α) for GPCM and GRM models.

delta1 First step parameter (δ_1) for the GPCM model.

delta2 Second step parameter (δ_2) for the GPCM model.

delta3 Third step parameter (δ_3) for the GPCM model.

beta1 First threshold parameter (β_1) for the GRM model.

beta2 Second threshold parameter (β_2) for the GRM model.

content Content category classification of the item.

time Item time.

group Item group identifier.

References

Luo, X. (2019). *Rmst: Computerized Adaptive Multistage Testing*. R package version 0.0.3. doi:10.32614/CRAN.package.Rmst. `https://CRAN.R-project.org/package=Rmst`

Examples

```
data(Rmst_pool)
```

`single_obj` *Maximize or Minimize a Single Objective Term*

Description

This function compiles a **single-objective optimization problem** for MST assembly. The optimization is to maximize or minimize a single linear score.

$$\max y$$

or

$$\min y$$

where $y = a^\top x$ is the linear objective term created by `objective_term()`.

Usage

```
single_obj(x, single_term)
```

Arguments

`x`	An object of class `mstATA_design` created by `mst_design()`.
`single_term`	A single objective term created by `objective_term()`. This function only accepts **one** objective term; supplying multiple terms will result in an error.

Details

1. Overview

There is only one objective term, representing a linear score of the form $a^\top x$, constructed by `objective_term()`.

Key characteristics for the objective term:

(1) Categorical or Quantitative

- **maximize or minimize the number of items belonging to a specific categorical level**, or
- **maximize or minimize the sum of quantitative item-level attribute values**, such as difficulty, discrimination, or item information function values.

(2) Module-, Pathway- or Panel-level

- **"Module-level"** – only items in a specified module contribute to the objective;
- **"Pathway-level"** – items in all modules belonging to one pathway contribute; and
- **"Panel-level"** – all item selections in the panel contribute.

(3) Relative or Absolute Objective

- **Relative objective**: maximize/minimize $a^\top x$.
- **Absolute deviation objective**: minimize the deviation from a specified target: $|a^\top x - g|$, where g is the target value.

Examples include:

- *Relative objective* – "Maximize the number of items in category c selected in Module 1."
- *Absolute objective* – "Minimize the deviation between the test information at $\theta = 0$ and the target value 15."

Value

A list of class `"compiled_objective"`

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse constraint matrix (class `"Matrix"`) for binary decision variables. Rows correspond to constraints; columns correspond to binary decision variables.

A_real A sparse constraint matrix (class `"Matrix"`) for continuous decision variables. Must have the same number of rows as `A_binary`.

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary A numeric vector of objective coefficients for binary decision variables.

C_real A numeric vector of objective coefficients for continuous decision variables.

sense "min" or "max"

decisionvar_name_new Character vector indicating the names of new decision variables, same length as `ncol(A_real)`.

decisionvar_type_new A character vector of `"C"` (continuous), same length as `ncol(A_real)`.

notes A list containing the strategy name, strategy arguments, and `y_bounds`.

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Let V_c^{item} denote the set of items that belong to category c and q_{i_s} be the quantitative attribute value of item i_s.

For absolute-deviation objectives, let T be the target value and d be a non-negative deviation variable.

Suppose there are M modules in an MST panel, Let $m = 1, \ldots, M$ denote the module id.

1. Module-level Objective (Module m):

(1a) Categorical attribute – relative objective

Specification: maximize or minimize the number of items from category c in module m.

$$\max / \min \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m}.$$

(1b) Quantitative attribute – relative objective

Specification: maximize or minimize the sum of item-level quantitative attribute values in module m.

$$\max / \min \sum_{s=1}^{S} \sum_{i_s} q_{i_s} x_{i_s,m}.$$

(1c) Categorical attribute – absolute deviation objective

Specification: minimize deviation between the number of items from category c in module m and the target value T.

$$\min d$$

subject to

$$\sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} - T \leq d$$

$$T - \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} \leq d$$

(1d) Quantitative attribute – absolute deviation objective

Specification: minimize deviation between the sum of item-level attribute values and the target value T.

$$\min d$$

subject to

$$\sum_{s=1}^{S} \sum_{i_s} q_{i_s} x_{i_s,m} - T \leq d$$

$$T - \sum_{s=1}^{S} \sum_{i_s} q_{i_s} x_{i_s,m} \leq d$$

2. Pathway-level Objective (Pathway r):

Let $\mathcal{M}$ denote the set of modules belonging to pathway r.

(2a) Categorical attribute – relative objective

$$\max / \min \sum_{m \in \mathcal{M}(r)} \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m}$$

(2b) Quantitative attribute – relative objective

$$\max / \min \sum_{m \in \mathcal{M}(r)} \sum_{s=1}^{S} \sum_{i_s} q_{i_s} x_{i_s,m}$$

(2c) Categorical attribute – absolute deviation objective

$$\min d$$

subject to

$$\sum_{m \in \mathcal{M}(r)} \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} - T \leq d$$

$$T - \sum_{m \in \mathcal{M}(r)} \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} \leq d$$

(2d) Quantitative attribute – absolute deviation objective

$$\min d$$

subject to

$$\sum_{m \in \mathcal{M}(r)} \sum_{s=1}^{S} \sum_{i_s} q_{i_s} x_{i_s,m} - T \leq d,$$

$$T - \sum_{m \in \mathcal{M}(r)} \sum_{s=1}^{S} \sum_{i_s} q_{i_s} x_{i_s,m} \leq d,$$

3. Panel-level Objective:

(3a) Categorical attribute – relative objective

$$\max / \min \sum_{m=1}^{M} \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m}.$$

(3b) Quantitative attribute – relative objective

$$\max / \min \sum_{m=1}^{M} \sum_{s=1}^{S} \sum_{i_s} q_{i_s} x_{i_s,m}.$$

(3c) Categorical attribute – absolute deviation objective

$$\min d$$

subject to

$$\sum_{m=1}^{M}\sum_{s=1}^{S}\sum_{i_s\in V_c^{item}} x_{i_s,m} - T \le d,$$

$$T - \sum_{m=1}^{M}\sum_{s=1}^{S}\sum_{i_s\in V_c^{item}} x_{i_s,m} \le d,$$

(3d) Quantitative attribute – absolute deviation objective

$$\min d$$

subject to

$$\sum_{m=1}^{M}\sum_{s=1}^{S}\sum_{i_s} q_{i_s} x_{i_s,m} - T \le d,$$

$$T - \sum_{m=1}^{M}\sum_{s=1}^{S}\sum_{i_s} q_{i_s} x_{i_s,m} \le d,$$

Let V_c^{stim} denote the set of stimuli that belong to category c and q_s be the quantitative attribute value of stimulus s.

The Module-/Pathway-/Panel level requirements can also be related with stimulus-level categorical/quantitative requirements.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool        = mini_itempool,
  design          = "1-3-3",
  exclude_pathway = c("1-1-3", "1-3-1"),
  pathway_length  = 8
)

# Example 1: relative objective for information at theta = 0 in the routing module
single_term <- objective_term(
  x             = test_mstATA,
  attribute     = "iif(theta=0)",
  cat_level         = NULL,
  applied_level = "Module-level",
  which_module  = 1,
  sense         = "max"
)
single_obj(x = test_mstATA, single_term = single_term)

# Example 2: absolute objective for information at theta = 0 in the routing module
single_term <- objective_term(
  x             = test_mstATA,
  attribute     = "iif(theta=0)",
```

```
  cat_level         = NULL,
  applied_level = "Module-level",
  which_module  = 1,
  sense         = "min",
  goal = 10
)
single_obj(x = test_mstATA, single_term = single_term)
```

`solution_itemcat_con` *Generate Solution-Level Constraints for the Min/Exact/Max Number of Unique Items from Specific Categorical Levels*

Description

This function constructs **solution-level constraints** that regulate the number of *unique items*–across all panels–that belong to specified categorical levels.

This constraint may enforce:

- a minimum number of unique items in category c,
- a maximum number of unique items in category c, or
- an exact number of unique items in category c

across the entire multi-panel MST solution.

Constraints are evaluated at the **Solution-level**, meaning that an item is considered 'selected' if it appears in *any* module of *any* panel.

The total number of constraints equals the number of categories in `cat_levels`.

Usage

```
solution_itemcat_con(x, attribute, cat_levels, operator, target_num)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` containing the **item categorical attribute**.
`cat_levels`	A character string or vector of category levels to constrain. These must be present in the unique values of the attribute column.
`operator`	A character string specifying the constraint type. Must be one of `"<="`, `"="`, or `">="`.
`target_num`	A scalar or vector giving the required number of unique items for each category level in `cat_levels`.

Details

1. Specification

The constraint enforces:

A minimum, exact, or maximum number of unique items from specified categories must be selected across all assembled panels.

Key characteristics:

- The attribute is type is *categorical*.
- The attribute is defined at the *item level* in the item pool.
- The constraint is applied at the **Solution-level**, i.e., across all panels.

Users may limit the constraint to a subset of categories using `cat_levels`.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

For each item i_s, s_{i_s} if item i_s is used in any module in any panel. s_{i_s} otherwise.

The solution-level variables track whether an item appears anywhere in the assembled multi-panel solution.

Let V_c^{item} denote the set of items that belong to category c.

$$\sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} s_{i_s} \lesseqgtr n_c^{item}.$$

Here:

- s_{i_s} is the binary decision variable indicating whether item i_s is used at least once across all panels.
- $\substack{\leq \\ = \\ \geq}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- n_c^{item} is the required number of unique item from category c across all panels.

Examples

```
data("mini_itempool")
test_mstATA <- mst_design(
  itempool      = mini_itempool,
  design        = "1-3-3",
  module_length = c(4,2,2,2,2,2,2)
)

# Require at least 2 unique content1 items and at least 6 unique content3 items
solution_itemcat_con(
  x        = test_mstATA,
  attribute = "content",
  cat_levels    = c("content1", "content3"),
  operator  = ">=",
  target_num = c(2, 6)
)
```

`solution_itemcount_con` *Generate Solution-Level Constraints for the Number of Unique Items Selected Across Panels*

Description

Constructs solution-level constraints on the *total number of distinct items* selected across all panels in a multistage test (MST) assembly.

This requirement enforces a user-specified:

- minimum number of unique items,
- maximum number of unique items, or
- exact (fixed) number of unique items

that may appear anywhere across all assembled panels.

These constraints operate at the **solution level**, using decision variables that indicate whether an item appears in *any module of any panel*.

The number of constraints generated is 1. If both a range needs to be specified, call this function twice.

Usage

```
solution_itemcount_con(x, operator, target_num)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().
operator	A character string specifying the constraint type. Must be one of "<=", "=", or ">=".
target_num	A scalar giving the required number of unique items across panels.

Details

1. Specification

This constraint enforces:

A minimum, exact, or maximum number of *unique items* must be selected across all panels in one solution.

Key characteristics:

- The attribute type is *categorical*: each item is assigned a unique ID.
- The attribute is defined at the *item level* in the item pool.
- The constraint is applied at the **Solution-level**, i.e., across all panels.

Value

An object of S3 class "mstATA_constraint" with named elements:

name A character vector indicating the specifications in each row of A_binary

specification A data.frame summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of A_binary.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

For each item i_s, s_{i_s} if item i_s is used in any module in any panel. s_{i_s} otherwise.

The solution-level variables track whether an item appears anywhere in the assembled multi-panel solution.

The total number of unique item selected at the solution level is controlled through:

$$\sum_{s=1}^{S}\sum_{i_s=1}^{I_s} s_{i_s} \overset{\leq}{\geq}{\underset{=}{}} n^{item}.$$

Here:

- s_{i_s} is the binary decision variable indicating whether item i_s is used at least once across all panels.
- $\overset{\leq}{\geq}{\underset{=}{}}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- n^{item} is the user-specified minimum, exact, or maximum number of unique items.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool      = mini_itempool,
  design        = "1-3",
  module_length = c(3,4,4,4)
)

# Require at least 20 unique items across all panels
solution_itemcount_con(test_mstATA, operator = ">=",target_num = 20)

# Require at least 20 at most 30 unique items across all panels
con1<-solution_itemcount_con(test_mstATA, operator = ">=",target_num = 20)
con2<-solution_itemcount_con(test_mstATA, operator = "<=",target_num = 30)
```

---

solution_stimcat_con *Generate Solution-Level Constraints for the Min/Exact/Max Number of Unique Stimuli in Specified Categories*

Description

This function constructs **solution-level constraints** that regulate the number of *unique stimuli*–across all panels–that belong to specified categorical levels.

The constraint may enforce:

- a minimum number of stimuli in category c,
- a maximum number of stimuli in category c, or
- an exact (equality) requirement in category c.

across the entire multi-panel MST solution.

Constraints are evaluated at the **Solution-level**, meaning that a stimulus is considered 'selected' if it appears in *any* module of *any* panel.

The total number of constraints equals the number of categories in `cat_levels`.

Usage

```
solution_stimcat_con(x, attribute, cat_levels, operator, target_num)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().
attribute	A string giving the column name in x$ItemPool containing **the stimulus categorical attribute**.
cat_levels	A character vector of category levels to be constrained. Must match the unique values in the specified attribute.
operator	A character string specifying the constraint direction: one of "<=", "=", or ">=".
target_num	A scalar or vector giving the required number of stimuli in each category listed in cat_levels.

Details

1. Specification

The constraint enforces:

A minimum, exact, or maximum number of unique stimuli belonging to specified categories must be selected across all panels in the multi-panel MST assembly.

Key characteristics:

- The attribute type is *categorical*.
- The attribute is defined at *stimulus level* in the item pool.
- The constraint is applied at the **Solution-level**, i.e., across all panels.

Users may limit the constraint to a subset of categories using cat_levels.

Value

An object of S3 class "mstATA_constraint" with named elements:

name A character vector indicating the specifications in each row of A_binary

specification A data.frame summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of A_binary.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

For each item i_s, s_{i_s} if item i_s is used in any module in any panel. s_{i_s} otherwise.

The solution-level variables track whether an item appears anywhere in the assembled multi-panel solution.

Let V_c^{stim} denote the set of stimuli that belong to category c.

$$\sum_{s \in V_c^{stim}} s_{i_s^*} \overset{\leq}{\geq} n_c^{stim}.$$

Here:

- $s_{i_s^*}$ indicates whether the pivot item for stimulus s is selected in any panel, thereby indicating whether the stimulus s is selected in any panel.
- $\overset{\leq}{\geq}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- n_c^{stim} is the required number of unique stimuli from category c across all panels.

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)

test_mstATA <- mst_design(
  itempool     = reading_itempool,
  design       = "1-3-3",
  module_length = c(14,12,12,12,12,12,12),
  pivot_stim_map = pivot_stim_map
)

# Require at least 3 history passages and 4 social studies passages
solution_stimcat_con(
  x             = test_mstATA,
  attribute     = "stimulus_type",
  cat_levels        = c("history", "social studies"),
  operator      = ">=",
  target_num    = c(3, 4)
)
```

`solution_stimcount_con` *Generate Solution-Level Constraints on the Number of Unique Stimuli Selected Across Panels*

Description

This function constructs a **solution-level constraint** that controls the number of *distinct stimuli* appearing in the final multi-panel MST assembly.

The constraint may impose a:

- minimum number of unique stimuli,
- maximum number of unique stimuli, or
- fixed (exact) number of unique stimuli.

that may appear anywhere across all assembled panels.

The number of constraints generated is 1. If both a range needs to be specified, call this function twice.

Usage

```
solution_stimcount_con(x, operator, target_num)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`operator`	A character string specifying the constraint type. Must be one of `"<="`, `"="`, or `">="`.
`target_num`	A scalar giving the required number of unique stimuli across panels.

Details

1. Specification

This constraint enforces:

A minimum, exact, or maximum number of unique stimuli must be selected across all panels in the assembled MST solution.

Key characteristics:

- The attribute type is *categorical*: each stimulus is assigned a unique ID.
- The attribute is defined at *stimulus level* in the item pool.
- The constraint is enforced at the **Solution-level**, i.e., across all panels.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

For each item i_s,

s_{i_s} if item i_s is used in any module in any panel. s_{i_s} otherwise.

The solution-level variables track whether an item appears anywhere in the assembled multi-panel solution.

The total number of unique stimuli selected at the solution level is controlled through:

$$\sum_{s=1}^{S-1} s_{i_s^*} \begin{matrix} \leq \\ \geq \\ = \end{matrix} n^{stim}.$$

Here:

- $s_{i_s^*}$ indicates whether the pivot item for stimulus s is selected in any panel, thereby indicating whether the stimulus s is selected in any panel.
- $\begin{matrix} \leq \\ \geq \\ = \end{matrix}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool = reading_itempool,
  stimulus = "stimulus",
  pivot_item = "pivot_item"
```

```
)

test_mstATA <- mst_design(
  itempool = reading_itempool,
  design = "1-3-3",
  module_length = c(5,7,7,7,8,8,8),
  pivot_stim_map = pivot_stim_map
)

# Require between 5 and 10 unique stimuli across all assembled panels
con1<-solution_stimcount_con(test_mstATA,">=",5)
con2<-solution_stimcount_con(test_mstATA,"<=",10)
```

`solve_model` *Solve an ATA Model Using a Mathematical Programming Solver*

Description

Solves an automated test assembly (ATA) model in the mstATA framework. The input model specification is optionally presolved to remove redundant rows and columns, checked for feasibility, and then solved using the requested solver.

Usage

```
solve_model(
  model_spec,
  continuous_bound_source = "compiled",
  continuous_bounds = NULL,
  solver = c("gurobi", "lpsolve", "HiGHS", "Symphony", "GLPK"),
  time_limit = 99999,
  check_feasibility = FALSE,
  show_progress = FALSE
)
```

Arguments

`model_spec` An assembly model specification of class `"mstATA_model"`, created by `onepanel_spec()` or `multipanel_spec()`.

`continuous_bound_source`

Character string specifying how lower and upper bounds for continuous decision variables are determined. Must be one of `"compiled"`, `"user"`, or `"none"`.

`"compiled"` Use bounds derived from compiled objective functions.

`"user"` Use user-specified bounds provided through `continuous_bounds`.

`"none"` Do not impose explicit bounds on continuous variables; lower and upper bounds are set to $-\infty$ and ∞, respectively.

The default is `"compiled"`.

`continuous_bounds`
: A named list specifying user-defined bounds for continuous decision variables when `continuous_bound_source = "user"`. The list must contain two numeric vectors:

  `lb` Lower bounds for continuous variables.

  `ub` Upper bounds for continuous variables.

  The length of each vector must equal the number of continuous variables (not slack variables) in the model. Each element of `lb` must be less than or equal to the corresponding element of `ub`. This argument must be `NULL` when `continuous_bound_source` is `"compiled"` or `"none"`.

`solver`
: A character string indicating which solver to use. One of `"gurobi"`, `"lpsolve"`, `"HiGHS"`, `"Symphony"`, or `"GLPK"`.

`time_limit`
: The maximum amount of computation time allocated to the solver. Default is 99999 seconds.

`check_feasibility`
: Logical. Default is `FALSE`. If `TRUE` and the HiGHS package is available, HiGHS is used to check whether the model is infeasible or unbounded before running the requested solver. If HiGHS certifies the model as `"INFEASIBLE"` or `"UNBOUNDED"`, the solving process terminates early and the result is returned immediately. If HiGHS does not certify feasibility within the time limit, the specified solver is invoked. This pre-check is purely diagnostic and does not affect the final solution when the model is feasible.

`show_progress`
: Logical. Whether to display solver progress information (e.g., incumbent solutions, node counts, and optimality gap) during optimization. Note that the level of detail depends on the underlying solver. Default is `FALSE`.

Value

A list with the following components:

solution A list containing the solver results with the following elements:

- **solution_found** A character string indicating the solver status (e.g., `"OPTIMAL"`, `"FEASIBLE"`, or `"INFEASIBLE"`, or `"UNBOUNDED"` or `"TIME_LIMIT"` or `"OTHER"`).
- **best_solution** A numeric vector of decision variable values corresponding to the best solution found.
- **objval** The objective function value associated with `best_solution`.
- **solution_list** Either `NULL` or a list of optimal solutions. If multiple optimal solutions exist, only the `"gurobi"` solver may return more than one solution.
- **check** Either `NULL` or a data frame summarizing constraint evaluation at the solution, including left-hand side values, constraint operators, right-hand side values, and residual values.
- **runtime** The total runtime of the solver in seconds.
- **iis** If `solver = "gurobi"` and model is infeasible, an irreducible inconsistent subsystem (IIS) will also be returned. It should be mentioned that an infeasible model may have multiple IISs. The one returned by Gurobi is not necessarily the smallest one; there may exist others with fewer constraints or bounds.

model An object of class `"mstATA_model"` representing the MILP formulation used by the solver. This includes the constraint matrix `A`, right-hand side `rhs`, constraint senses, objective coefficients, bounds, variable types, variable names, model sense, and row names.

solver A character string indicating the solver used (as requested by `solver`).

`solve_with_slack` *Solve An Infeasible mstATA Model Using Block-level Slack Variables*

Description

Attempts to recover feasibility for an infeasible `mstATA_model` by relaxing selected non-core constraint blocks using slack variables with user-specified penalties. A single continuous slack variable is introduced for each relaxable constraint block, allowing controlled violations while preserving the core structure of the MST design.

This function is intended as a diagnostic and fallback tool rather than a replacement for strict feasibility checks.

Usage

```
solve_with_slack(
  model_spec,
  cat_penalty,
  quant_penalty,
  solver = c("gurobi", "lpsolve", "HiGHS", "Symphony", "GLPK"),
  time_limit = 99999
)
```

Arguments

`model_spec`	An object of class `mstATA_model`. The model may be infeasible under its original specification.
`cat_penalty`	Numeric scalar giving the penalty coefficient applied to slack variables associated with categorical constraint blocks. Larger values discourage violations of categorical requirements.
`quant_penalty`	Numeric scalar giving the penalty coefficient applied to slack variables associated with quantitative constraint blocks.
`solver`	A character string indicating which solver to use. One of `"gurobi"`, `"lpsolve"`, `"HiGHS"`, `"Symphony"`, or `"GLPK"`.
`time_limit`	The maximum amount of computation time allocated to the solver. Default is 99999 seconds.

Details

The relaxed model is constructed as follows:

- **Hard constraints (never relaxed):**
  - Objective-related constraints.
  - Essential MST structure constraints (e.g., module- and pathway-level item count constraints, optional requirements for routing decision points).
  - Logical constraints (e.g., enemy relationships, item-stimulus grouping relationships, item exposure control).
  - Equality constraints, identified by `specification$Operator == "(exact)"` at the block level.
- **Soft constraints (relaxed with slack):**
  - Categorical constraint blocks.
  - Quantitative constraint blocks.

For each relaxable constraint block, one nonnegative continuous slack variable is introduced and shared across all rows in that block. Slack coefficients are assigned according to the row operator:

- For `"<="` rows, the slack enters with coefficient -1.
- For `">="` rows, the slack enters with coefficient $+1$.

Soft constraint blocks must contain only `"<="` or `">="` rows.

The objective function is augmented by adding penalty terms for each slack variable. Categorical slack variables are penalized **more heavily** than quantitative slack variables to reflect their higher priority.

Value

A solver result object returned by `solve_model()`, containing the optimized solution, solver status, and objective value for the relaxed model. Slack variable values indicate the magnitude of constraint violations at the block level.

Mathematical Formulation

Consider an infeasible mstATA model consisting of a set of constraint blocks indexed by $b = 1, \ldots, B$. Each block b corresponds to one row in the specification table and may contain multiple linear constraint rows indexed by $r \in \mathcal{R}$.

For each constraint row, let $a_r^\top x$ denote the left-hand side linear expression and d_r the right-hand side constant.

Soft constraint blocks and slack variables. For each relaxable constraint block b (categorical or quantitative), a single nonnegative continuous slack variable $s_b \geq 0$ is introduced and shared across all rows in that block.

Original (hard) constraints. For a row $r \in \mathcal{R}$, the original linear constraint takes one of the following forms:

- **Upper-bound constraint:**

$$a_r^\top x \leq d_r$$

- **Lower-bound constraint:**

$$a_r^\top x \geq d_r$$

Equality constraints ($a_r^\top x = d_r$) are not relaxable and therefore are excluded from slack construction.

Relaxed constraints with block-level slack. For a relaxable block b, each row $r \in \mathcal{R}$ is modified as follows:

- If the original row is $a_r^\top x \leq d_r$, the relaxed form is

$$a_r^\top x \leq d_r + s_b.$$

- If the original row is $a_r^\top x \geq d_r$, the relaxed form is

$$a_r^\top x \geq d_r - s_b.$$

Equivalently, in matrix form, the slack variable s_b enters each row with coefficient -1 for `"<="` constraints and $+1$ for `">="` constraints. A single slack variable therefore bounds the maximum violation across all rows in the block.

Categorical vs quantitative constraints. The mathematical relaxation is identical for categorical and quantitative constraint. They differ only in the penalty applied to their slack variables in the objective function. Categorical constraints are penalized more heavily to reflect higher priority.

Objective function with slack penalties. Let $\mathcal{B}_{cat}$ and $\mathcal{B}_{quant}$ denote the sets of categorical and quantitative soft constraint blocks, respectively. The relaxed optimization problem augments the original objective by adding slack penalties:

$$\min f(x) + \sum_{b \in \mathcal{B}_{cat}} \lambda_{cat}\, s_b + \sum_{b \in \mathcal{B}_{quant}} \lambda_{quant}\, s_b,$$

where $f(x)$ denotes the original objective function, and λ_{cat} are user-specified penalty coefficients.

If the original problem is a maximization problem, the equivalent formulation is obtained by subtracting the slack penalties from the objective:

$$\max f(x) - \sum_{b \in \mathcal{B}_{cat}} \lambda_{cat}\, s_b - \sum_{b \in \mathcal{B}_{quant}} \lambda_{quant}\, s_b.$$

The resulting solution represents a best-compromise feasible design that minimizes violations of high-priority constraints before lower-priority ones.

See Also

`solve_model()`

stimcat_con *Generate Stimulus-Level Constraints to Explicitly Select or Not Select Specific Stimuli in MST Panel Assembly*

Description

This function generates linear constraints that force specific stimuli to be either selected or not selected in the MST panel assembly.

This constraint can be enforced at the "Module-level", "Pathway-level", or "Panel-level".

The number of generated linear constraints depends on the application level:

- **Module-level:** when which_module is provided and which_pathway = NULL
  The number of constraints is **(number of stimuli in stim_ids) x (number of modules specified)**
- **Pathway-level:** when which_pathway is provided and which_module = NULL
  The number of constraints is **(number of stimuli in stim_ids) x (number of pathways specified)**
- **Panel-level:** both which_module and which_pathway are NULL
  The number of constraints is **(number of stimuli in stim_ids)**

Usage

```
stimcat_con(
  x,
  stim_ids,
  select = TRUE,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().
stim_ids	A numeric vector of stimuli indices or a character vector of stimuli names
select	Logical. Default is TRUE.
which_module	Integer vector of modules. Must be consistent with the choices made in: • test_stimcount_con() • test_stimcat_con() • test_stimquant_con()
which_pathway	Integer vector of pathways. Must be consistent with the choices made in: • test_stimcount_con() • test_stimcat_con() • test_stimquant_con()

Details

1. Specification

The constraint enforced by this function is:

The specified stimuli must be selected or not selected in the chosen set of modules, pathways, or across the entire panel.

The key properties of this constraint are:

- The attribute type is *categorical*: each stimulus has a unique ID and may be treated as an unique category.
- The constraint is defined at the *stimulus level* in the item pool (each stimulus is a distinct entity).
- The constraint can be applied at:
  - `"Module-level"` implies stimulus must be selected/not selected in the specified module(s).
  - `"Pathway-level"` implies stimulus must be selected/not selected in one of the modules composing that pathway.
  - `"Panel-level"` implies stimulus must be selected/not selected somewhere in the panel (but placement is not specified).

2. Interaction with `panel_itemreuse_con()`

When `overlap = FALSE` in `panel_itemreuse_con()`, each item can at most be selected once in a panel. This also indicates that each stimulus pivot item–may appear in at most one module in the panel.

Therefore:

- `which_module` or `which_pathway` must be a scalar (not a vector).
- If both are `NULL`, the constraint simply enforces that the stimulus must appear somewhere in the panel, without specifying the module/pathway.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A right-hand-side vector of 1s or 0s depending on whether `select` is `TRUE` or `FALSE`.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

1. Module-level stimulus selection/not selection

In specified module m: for all s (s= 1,2...,S-1) specified in `stim_ids`

$$x_{i_s^*,m} = 1(selected)$$

$$x_{i_s^*,m} = 0(notselected)$$

2. Pathway-level stimulus selection/not selection

In specified pathway r: for all s (s= 1,2...,S-1) specified in `stim_ids`

$$\sum_{m \in r} x_{i_s^*,m} = 1(selected)$$

$$\sum_{m \in r} x_{i_s^*,m} = 0(notselected)$$

3. Panel-level stimulus selection/not selection

In a panel: for all s (s= 1,2...,S-1) specified in `stim_ids`

$$\sum_{m} x_{i_s^*,m} = 1(selected)$$

$$\sum_{m} x_{i_s^*,m} = 0(notselected)$$

Here:

- $x_{i_s^*,m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus is selected in that module.
- $m \in r$ denote modules belonging to pathway r.

Examples

```
data("reading_itempool")
pivot_stim_map <- create_pivot_stimulus_map(
  itempool = reading_itempool,
  stimulus = "stimulus",
  pivot_item = "pivot_item"
)

test_mstATA <- mst_design(
```

```
  itempool = reading_itempool,
  design = "1-3-3",
  module_length = c(5,7,7,7,8,8,8),
  pivot_stim_map = pivot_stim_map
)

# Example 1: Force stimulus 1 to be selected in the routing module
stimcat_con(
  x = test_mstATA,
  stim_ids = 1,
  select = TRUE,
  which_module = 1
)

# Example 2: Force stimulus 1 to be included somewhere in the MST panel
stimcat_con(
  x = test_mstATA,
  stim_ids = 1,
  select = TRUE
)

# Example 3: Stimulus 1 must appear in stage-2 easy and medium modules
stimcat_con(
  x = test_mstATA,
  stim_ids = 1,
  select = TRUE,
  which_module = c(2,3)
)

# Example 4: Stimulus 1 must appear in the REE pathway
stimcat_con(
  x = test_mstATA,
  stim_ids = 1,
  select = TRUE,
  which_pathway = 1
)

# Example 5: Stimuli 1 and 3 must appear in the routing module
stimcat_con(
  x = test_mstATA,
  stim_ids = c(1,3),
  select = TRUE,
  which_module = 1
)

# Example 6: Stimuli 1 and 3 must appear in the REE pathway
stimcat_con(
  x = test_mstATA,
  stim_ids = c(1,3),
  select = TRUE,
  which_pathway = 1
)
```

```
# Example 7: Stimuli 1 and 3 must appear somewhere in the panel
stimcat_con(
  x = test_mstATA,
  stim_ids = c(1,3),
  select = TRUE
)
```

`stimquant_con` *Generate Stimulus-Level Constraints Requiring Stimulus-Level Quantitative Attributes to Be Greater Than, Less Than, or Within a Specified Range*

Description

This function generates linear constraints ensuring that each selected stimulus satisfies a specified quantitative requirement (e.g., stimulus word count, stimulus difficulty, readability index).

This constraint can be enforced at the `"Module-level"`, `"Pathway-level"`, or `"Panel-level"`. It should be mentioned that the formulation requires stimulus-level attributes to be strictly positive.

The number of generated linear constraints depends on the application level:

- **Module-level:** when `which_module` is provided and `which_pathway = NULL`
  The number of constraints is **(number of stimuli) x (number of modules specified) x side**
- **Pathway-level:** when `which_pathway` is provided and `which_module = NULL`
  The number of constraints is **(number of stimuli) x (number of unique modules in specified pathways) x side**
- **Panel-level:** both `which_module` and `which_pathway` are `NULL`
  The number of constraints is **(number of stimuli) x (total number of modules) x side**

where `side = 1` for one-sided constraints (min-only or max-only) and `side = 2` when both `min` and `max` are provided and min is not equal to max.

Usage

```
stimquant_con(
  x,
  attribute,
  min = NULL,
  max = NULL,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that represents the quantitative attribute for the stimulus.
`min`	A numeric scalar for the lower bound.
`max`	A numeric scalar for the upper bound.
`which_module`	Integer vector of modules. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`
`which_pathway`	Integer vector of pathways. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`

Details

1. Specification

The constraint enforced is:

Every selected stimulus must satisfy the required stimulus-level quantitative attribute bound(s).

- The attribute type is *quantitative*.
- The attribute is defined at the *stimulus level* in the item pool.
- Application levels:
  - **Module-level**: enforce stimuli selected in specific module(s) satisfy the quantitative requirement.
  - **Pathway-level**: enforce stimuli selected in any module of a pathway satisfy the quantitative requirement.
  - **Panel-level**: enforce stimuli selected in the panel satisfy the quantitative requirement.

2. Important Restriction

This function assumes that all stimulus-level quantitative attributes q_s are **strictly positive**.

If stimulus attributes may be zero or negative (e.g., stimulus-level difficulty, centered difficulty values), this constraint formulation does not correctly enforce quantitative bounds. In such cases, users should instead apply stimulus-level selection logic using `item_module_eligibility()` inside mst_design(), which filters eligible stimuli (and linked items) before assembly.

For example, to require all routing-module stimuli to have text feature scores between -1 and 1, users should filter stimuli into the routing module eligibility set using `item_module_eligibility()`.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A right-hand-side vector of 1s or 0s depending on whether `select` is `TRUE` or `FALSE`.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let q_s be the quantitative attribute value of stimulus s.

One-sided upper-bound constraint:

$$q_s \, x_{i_s^*,m} \leq b_q^{stim,max}, s = 1, \ldots, S-1.$$

One-sided lower-bound constraint:

$$b_q^{stim,min} \, x_{i_s^*,m} \leq q_s, s = 1, \ldots, S-1.$$

For two-sided range constraints, both inequalities are included. (`min` and `max` both provided and not equal), the number of constraints is doubled (`side = 2`). For one-sided constraints, `side = 1`.

Here:

- $x_{i_s^*,m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus s is selected in that module.
- q_s (**must be positive values**) denote the values of the quantitative attribute for stimulus s.
- $b_q^{stim,min}$ and $b_q^{stim,max}$ are the lower and upper allowable bounds for the attribute q_s.

See Also

`mst_design()`

Examples

```
data("reading_itempool")
pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)

test_mstATA <- mst_design(
  itempool      = reading_itempool,
  design        = "1-3-3",
  module_length = c(14, 12, 12, 12, 12, 12, 12),
  pivot_stim_map = pivot_stim_map
)

# Every selected stimulus in the routing module has 80-110 words.
stimquant_con(
  x = test_mstATA,
  attribute = "stimulus_words",
  min = 80, max = 110, which_module =1
)
```

`stim_itemcat_con` *Generate Itemset-Level Constraints for Minimum, Exact, or Maximum Numbers of Stimulus-Linked Items from Specific Categorical Levels*

Description

This function constructs linear constraints ensuring that, for every selected stimulus, the items associated with that stimulus meet the required minimum, exact, or maximum counts from specified categorical levels (e.g., content area, skill type, item format).

The total number of generated linear constraints =

(number of stimuli) x (number of category levels) x (number of involved modules)

For a range constraint (i.e., both a minimum and a maximum number of items requirement), the function must be called twice:

- once with `target_num` = `min` and `operator` = `">="`, and
- once with `target_num` = `max` and `operator` = `"<="`.

Usage

```
stim_itemcat_con(
  x,
  attribute,
  cat_levels,
  operator,
  target_num,
```

```
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that represents the **item-level categorical attribute**.
`cat_levels`	Character vector of category levels to constrain. Must match values in `attribute`.
`operator`	A character string specifying the inequality or equality operator. Must be one of `"<="`, `"="`, or `">="`.
`target_num`	A scalar/vector specifying the required number of items for the category levels in `cat_levels`.
`which_module`	Integer vector of modules. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`
`which_pathway`	Integer vector of pathways. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`

Details

The constraint enforced is:

If a stimulus is selected, a minimum, exact, or maximum number of its linked items must belong to the specified category level.

The key properties of this constraint are:

- The attribute type is *logical*: a conditional "if-then" relationship between the selection of a stimulus and the required selection of its associated items.
- The attribute is defined at the *itemset level: set size for each stimulus* in the item pool through the mapping between a stimulus and the items that belong to it.
- *Conditional* stimulus-item constraints (such as stim_itemcount_con(), stim_itemcat_con(), stim_itemquant_con()) must be applied at the `"Module-level"` because **items linked to a selected stimulus cannot be distributed across multiple modules**: if the stimulus is selected in a module, all items required from that stimulus must appear in the same module.
- **Important:** The arguments `which_module` and `which_pathway` must be consistent with the choices made in test_stimcount_con(), test_stimcat_con(), and test_stimquant_con().

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_c^{item} denote the set of items that belong to category c.

$$n_c^{\min} x_{i_s^*,m} \leq \sum_{i_s \in V_c^{item}} x_{i_s,m} \leq n_c^{\max} x_{i_s^*,m}, s = 1, \ldots, S-1.$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $x_{i_s^*,m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus s is selected in that module.
- $n_c^{\min}$ and $n_c^{\max}$ are the required minimum and maximum number of stimulus-based items from category level c if stimulus s is selected.

See Also

`stim_itemcount_con()`,

`stim_itemquant_con()`

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)

test_mstATA <- mst_design(
  itempool      = reading_itempool,
  design        = "1-3-3",
  module_length = c(14,12,12,12,12,12,12),
  pivot_stim_map = pivot_stim_map
)

# Example 1: At least 2 MC and at least 1 TEI item must be selected when the stimulus is selected.
stim_itemcat_con(
  x = test_mstATA,
  attribute = "itemtype",
  cat_levels = c("MC","TEI"),
  operator = ">=",
  target_num = c(2,1)
)

# Example 2: At most 5 MC and at most 3 TEI items may be selected from each stimulus.
stim_itemcat_con(
  x = test_mstATA,
  attribute = "itemtype",
  cat_levels = c("MC","TEI"),
  operator = "<=",
  target_num = c(5,3)
)

# Example 3: Exactly 4 MC and 2 TEI items must be selected from each stimulus.
stim_itemcat_con(
  x = test_mstATA,
  attribute = "itemtype",
  cat_levels = c("MC","TEI"),
  operator = "=",
  target_num = c(4,2)
)
```

---

stim_itemcount_con *Generate Itemset-Level Constraints for the Number of Selected Items*

Description

This function constructs linear constraints that enforce minimum, exact, or maximum counts on the number of items associated with a selected stimulus. It supports both full selection (all items linked

to a stimulus must be selected if the stimulus is selected) and partial selection (only a subset of linked items must be selected).

The total number of generated linear constraints depends on the provided values of `min` and `max`.

- **Full selection (min = NULL and max = NULL):** All items linked to a stimulus must either all be selected or all not be selected.

  The number of constraints is **2 x (number of stimuli) x (number of involved modules)**.

  For each stimulus, the lower bound and the upper bound is the number of items linked to the stimulus.

- **Only max is specified:** An upper bound is imposed, but no lower bound.

  The number of constraints is **(number of stimuli) x (number of involved modules)**.

- **Only min is specified:** A lower bound is imposed; the upper bound defaults to the total number of items belonging to that stimulus.

  The number of constraints is **2 x (number of stimuli) x (number of involved modules)**.

- **Both min and max are specified:**

  The number of constraints is **2 x (number of stimuli) x (number of invovle modules)**.

Usage

```
stim_itemcount_con(
  x,
  min = NULL,
  max = NULL,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`min`	An optional scalar specifying the minimum number of items to be included conditional on the selection of its stimulus.
`max`	An optional scalar specifying the maximum number of items to be included conditional on the selection of its stimulus.
`which_module`	Integer vector of modules. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`
`which_pathway`	Integer vector of pathways. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`

Details

The constraint enforced is:

If a stimulus is selected, then a minimum, exact, or maximum number of items linked to that stimulus must also be selected.

The key properties of this constraint are:

- The attribute type is *logical*: a conditional "if-then" relationship between the selection of a stimulus and the required selection of its associated items.
- The attribute is defined at the *itemset level: set size for each stimulus* in the item pool through the mapping between a stimulus and the items that belong to it.
- *Conditional* stimulus-item constraints (such as stim_itemcount_con(), stim_itemcat_con(), stim_itemquant_con()) must be applied at the `"Module-level"` because **items linked to a selected stimulus cannot be distributed across multiple modules**: if the stimulus is selected in a module, all items required from that stimulus must appear in the same module.
- **Important:** The arguments `which_module` and `which_pathway` must be consistent with the choices made in test_stimcount_con(), test_stimcat_con(), and test_stimquant_con().

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Partial selection with min and max constraints

$$n^{\min} x_{i_s^*,m} \leq \sum_{i_s=1}^{I_s} x_{i_s,m} \leq n^{\max} x_{i_s^*,m}, s = 1, \ldots, S-1.$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $x_{i_s^*,m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus is selected in that module.
- $n^{\min}$ and $n^{\max}$ specify the minimum and maximum allowable number of items that must be selected if its stimulus is selected.

All-in/All-out selection:

$$n^s x_{i_s^*,m} \leq \sum_{i_s=1}^{I_s} x_{i_s,m} \leq n^s x_{i_s^*,m}, s = 1, \ldots, S-1.$$

where n^s is the number of items linked to stimulus s.

Partial selection with min constraints

$$n^{\min} x_{i_s^*,m} \leq \sum_{i_s=1}^{I_s} x_{i_s,m} \leq n^s x_{i_s^*,m}, s = 1, \ldots, S-1.$$

where $n^{\min}$ specifies the minimum number of items that must be selected if its stimulus is selected. n^s is the number of items linked to stimulus s. The requirement that a maximum number of items that must be selected if its stimulus is selected is automatically added. This produces a safe gating constraint: if the pivot item is not selected (meaning the stimulus is not selected), then $x_{i_s^*,m} = 0$ forces all corresponding item-selection variables $x_{i_s,m} = 0$, ensuring that no items from an unselected stimulus can be selected.

Partial selection with max constraints

$$\sum_{i_s=1}^{I_s} x_{i_s,m} \leq n^{\max} x_{i_s^*,m}, s = 1, \ldots, S-1.$$

where $n^{\max}$ specifies the minimum and maximum number of items that must be selected if its stimulus is selected. When the pivot item is not selected (meaning the stimulus is not selected), $x_{i_s^*,m} = 0$ forces all corresponding item-selection variables $x_{i_s,m} = 0$, ensuring that no items from an unselected stimulus can be selected.

See Also

stim_itemcat_con(),

stim_itemquant_con()

Examples

```
data("reading_itempool")
pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)
```

```
test_mstATA <- mst_design(
  itempool      = reading_itempool,
  design        = "1-3-3",
  module_length = c(14, 12, 12, 12, 12, 12, 12),
  pivot_stim_map = pivot_stim_map
)

# Example 1: Full selection.
# If a stimulus is selected, all items linked to that stimulus must also be selected.
stim_itemcount_con(x = test_mstATA)

# Example 2: Range selection (min and max).
# If a stimulus is selected, between 2 and 5 of its linked items must be selected.
stim_itemcount_con(
  x = test_mstATA,
  min = 2,
  max = 5
)

# Example 3: Minimum-only selection.
# If a stimulus is selected, at least 5 of its items must be selected.
# Upper bound defaults to total items per stimulus
stim_itemcount_con(
  x = test_mstATA,
  min = 5
)

# Example 4: Maximum-only selection.
# If a stimulus is selected, at most 8 of its items may be selected.
# Only upper bound active
stim_itemcount_con(
  x = test_mstATA,
  max = 8
)

# Example 5: Exact selection.
# If a stimulus is selected, exactly 6 of its items must be selected (min = max = 6).
stim_itemcount_con(
  x = test_mstATA,
  min = 6,
  max = 6
)
```

`stim_itemquant_con` *Generate Itemset-Level Constraints for Minimum/Exact/Maximum Sum of Item Quantitative Attribute Values*

Description

Creates a conditional quantitative constraint requiring that, if a stimulus is selected, the *sum of quantitative attribute values* of the items linked to that stimulus must satisfy a user-specified lower

bound, upper bound, or exact value.

This function enforces a logical "if-then" relationship between stimulus selection and the quantitative properties (difficulty, response time) of its associated items.

The total number of generated linear constraints = (number of stimuli) x (number of involved modules)

For a range constraint (i.e., both a minimum and a maximum quantitative requirement), the function must be called twice:

- once with `target_value = min` and `operator = ">="`, and
- once with `target_value = max` and `operator = "<="`.

Usage

```
stim_itemquant_con(
  x,
  attribute,
  operator,
  target_value,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that represents the **item-level quantitative attribute**.
`operator`	A character string of constraint operators, one of `"<="`, `"="`, or `">="`.
`target_value`	Target values in specific module/pathway (`which_module`, `which_pathway`).
`which_module`	Integer vector of modules. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`
`which_pathway`	Integer vector of pathways. Must be consistent with the choices made in: • `test_stimcount_con()` • `test_stimcat_con()` • `test_stimquant_con()`

Details

The constraint enforced is:

If a stimulus is selected, then the items linked to it must collectively satisfy a minimum, maximum, or exact sum of a quantitative item attribute.

The key properties of this constraint are:

- The attribute type is *logical*: a conditional "if-then" relationship between the selection of a stimulus and the required selection of its associated items.
- The attribute is defined at the *itemset level: set size for each stimulus* in the item pool through the mapping between a stimulus and the items that belong to it.
- *Conditional* stimulus-item constraints (such as stim_itemcount_con(), stim_itemcat_con(), stim_itemquant_con()) must be applied at the `"Module-level"` because **items linked to a selected stimulus cannot be distributed across multiple modules**: if the stimulus is selected in a module, all items required from that stimulus must appear in the same module.
- **Important:** The arguments `which_module` and `which_pathway` must be consistent with the choices made in test_stimcount_con(), test_stimcat_con(), and test_stimquant_con().

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let q_{i_s} be the quantitative attribute value of item i_s. Then for module m, the conditional constraint is:

$$\sum_{i_s=1}^{I_s} q_{i_s} x_{i_s,m} \lessgtr b_q^m x_{i_s^*,m}, s = 1, \ldots, S-1.$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $x_{i_s^*,m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus s is selected in that module.
- b_q is the required quantitative bound for the sum of item-level quantitative attribute linked to every stimulus in module m (minimum, maximum, or exact value).

See Also

stim_itemcat_con(),

stim_itemcount_con()

Examples

```
data("reading_itempool")
pivot_stim_map<-create_pivot_stimulus_map(itempool = reading_itempool,
                                          stimulus = "stimulus",
                                          pivot_item = "pivot_item")
# Example 1: if a stimulus is chosen, the sum difficulty of its linked items
# may be required to be smaller than 0.5.
test_mstATA<-mst_design(itempool = reading_itempool,design = "1-3-3",
                        module_length = c(14,12,12,12,12,12,12),
                       pivot_stim_map = pivot_stim_map)
stim_itemquant_con(x = test_mstATA,attribute = "difficulty",
                   operator = "<=",target_value = 0.5,
                  which_module = NULL,which_pathway = NULL)
```

TD123_panel *Precomputed Top-Down MST Assembly Example (TD123)*

Description

A precomputed MST panel assembled using the top-down strategy with the HiGHS solver. Included to avoid long runtime during vignette building.

Usage

```
TD123_panel
```

Format

An object of class mstATA_panel.

Source

Generated using mstATA::assembled_panels() and mstATA::solve_model()with the HiGHS solver.

TD12_panel — *Precomputed Top-Down MST Assembly Example (TD12)*

Description

A precomputed MST panel assembled using the top-down strategy with the HiGHS solver demonstrated in the *Three ATA Strategies* Vignette.

Usage

```
TD12_panel
```

Format

An object of class `mstATA_panel`.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

test_itemcat_con — *Generate Module- or Pathway-Level Constraints for the Minimum, Exact, or Maximum Number of Items from Specific Categorical Levels*

Description

This function generates a linear constraint matrix enforcing a minimum, exact, or maximum number of items from specified categorical levels (e.g., content area, item format).

Constraints may apply at the:

- **"Module-level"** – each module is treated as an independent test form
- **"Pathway-level"** – each pathway is treated as an independent test form

The total number of constraints depends on the application level and the number of category levels specified in `cat_levels`:

- **Module-level:** when `which_module` is provided OR both `which_module` and `which_pathway` are `NULL`

  The number of constraints is **(number of category levels)x (number of modules specified)**
- **Pathway-level:** when `which_pathway` is provided and `which_module` = `NULL`

  The number of constraints is **(number of category levels) x (number of pathways specified)**

Users may constrain a subset of modules or pathways using the `which_module` or `which_pathway` arguments, and may restrict the constraint to specific categories using `cat_levels`.

Usage

```
test_itemcat_con(
  x,
  attribute,
  cat_levels,
  operator,
  target_num,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().
attribute	A string giving the column name in x$ItemPool that contains the **item categorical attribute**.
cat_levels	A character string or vector of categorical levels to be constrained.
operator	A character string specifying the inequality or equality operator. Must be one of "<=", "=", or ">=". For range-based constraints, use test_itemcat_range_con() instead.
target_num	A matrix or scalar/vector specifying the required number of items for the category levels in cat_levels. If scalar/vector is provided, it expands to a matrix. See **Details**.
which_module	Optional integer vector of module indices to which the constraints apply.
which_pathway	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforces:

The test must meet a minimum, exact, or maximum number of items from the specified categories.

Key characteristics:

- The attribute type is *categorical*.
- The attribute is defined at the *item level* in the item pool.
- The constraints are applied at either **"Module-level"** or **"Pathway-level"**.

2. target_num

The constraint level is determined by which selection argument is provided:

- **Module-level**:
  - Requires the minimum/exact/maximum number of items from specified category levels to be selected in specified module.

  - Expanded `target_num` matrix: each row = one specified module, each column = one specified category.

- **Pathway-level**:
  - Requires the minimum/exact/maximum number of items from specified category levels to be selected across the modules belonging to specified pathway.
  - Expanded `target_num` matrix: each row = one specified pathway, each column = one specified category.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_c^{item} denote the set of items that belong to category c.

1. Module-level constraint (for module m)

$$\sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} \overset{\leq}{\geq} n_c^m$$

2. Pathway-level constraint (for pathway r)

$$\sum_{m \in r} \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} \overset{\leq}{\geq} n_c^r$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.
- The stacked operator, $\begin{matrix}\leq \\ = \\ \geq\end{matrix}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- n_c^m specify the required number of items from category level c in module m.
- n_c^r specify the required number of items from category level c in pathway r.

Examples

```
data("mini_itempool")

test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  module_length = c(4,2,2,2,2,2,2)
)
# Example 1: category constraints across modules in the same stage
con1<-test_itemcat_con(x = test_mstATA,attribute = "itemtype",cat_levels = c("MC","TEI"),
                                       operator = "=",target_num = c(3,1),
                                       which_module = 1,which_pathway = NULL)
con2<-test_itemcat_con(x = test_mstATA,attribute = "itemtype",cat_levels = c("MC","TEI"),
                                       operator = "=",target_num = c(2,0),
                                       which_module = 2:4,which_pathway = NULL)
con3<-test_itemcat_con(x = test_mstATA,attribute = "itemtype",cat_levels = c("MC","TEI"),
                                       operator = "=",target_num = c(2,0),
                                       which_module = 5:7,which_pathway = NULL)
# Example 2: category constraints per pathway
test_mstATA<-mst_design(itempool = mini_itempool,design = "1-3-3",
                        exclude_pathways = c("1-1-3","1-3-1"),pathway_length = 8)
test_itemcat_con(x = test_mstATA,
                           attribute ="itemtype",cat_levels = c("MC","TEI"),
                           operator = "=",
                           target_num = c(7,1),
                           which_pathway = 1:7)

# Example 3: constraints on combinations of categorical attributes
mini_itempool$content_itemtype <-
  paste(mini_itempool$content, mini_itempool$itemtype)

test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  exclude_pathways = c("1-1-3","1-3-1"),
  pathway_length = 8
)
test_itemcat_con(
  x = test_mstATA,
  attribute = "content_itemtype",
  cat_levels = paste0("content", rep(1:4, each = 2)," ", c("MC","TEI")),
  operator = "=",
  target_num = c(1,1, 1,1, 1,0, 2,1),
```

```
  which_pathway = 1:7
)
```

`test_itemcat_range_con` *Generate Module- or Pathway-Level Range Constraints for the Number of Items from Specific Categorical Levels*

Description

This function generates constraint matrices that enforce lower and upper bounds on the number of selected items belonging to specific categorical attribute levels (e.g., content area, item format) within modules or pathways.

It supports two specification formats:

- **Direct range constraints:** `min` and/or `max`
- **Target plus or minus deviation constraints:** `target` and `deviation`

Constraints may apply at the:

- **"Module-level"** – each module is treated as an independent test form
- **"Pathway-level"** – each pathway is treated as an independent test form

The total number of constraints depends on the application level and the number of category levels in `cat_levels`:

- **Module-level:** when `which_module` is provided, or both `which_module` and `which_pathway` are `NULL`
  The number of constraints = **(number of category levels) x (number of modules specified) x side**
- **Pathway-level:** when `which_pathway` is provided and `which_module = NULL`
  The number of constraints = **(number of category levels) x (number of pathways specified) x side**

Here, `side = 2` for range constraints (lower + upper bound) and `side = 1` when only one bound is active.

Usage

```
test_itemcat_range_con(
  x,
  attribute,
  cat_levels,
  min = NULL,
  max = NULL,
  target = NULL,
  deviation = NULL,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that contains the **item categorical attribute**.
`cat_levels`	Character vector of category levels to be constrained.
`min`	Optional minimum number of items (scalar, vector, or matrix).
`max`	Optional maximum number of items (scalar, vector, or matrix).
`target`	Optional target number of items (scalar, vector, or matrix).
`deviation`	Optional allowable deviation from `target` (scalar, vector, or matrix).
`which_module`	Optional integer vector of module indices to which the constraints apply.
`which_pathway`	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforces:

The number of selected items in each specified category must fall within an allowable range.

Key characteristics:

- The attribute type is *categorical*.
- The attribute is defined at the *item level* in the item pool.
- The constraints are applied at either **"Module-level"** or **"Pathway-level"**.

2. Range specification: min, max, target, deviation

Each of the four range-specification arguments may be: `NULL`, scalar, vector, or matrix.

Interpretation:

- `min`: minimum number of items per category per module/pathway
- `max`: maximum number of items per category per module/pathway
- `target`: desired number of items per category per module/pathway
- `deviation`: allowable deviation from `target`

When only `target` and `deviation` are provided: min = target - deviation. max = target + deviation.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A string indicating the specification name

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_c^{item} denote the set of items that belong to category c.

1. Module-level constraint (for module m)

$$n_c^{m,\min} \leq \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} \leq n_c^{m,\max}$$

2. Pathway-level constraint (for pathway r)

$$n_c^{r,\min} \leq \sum_{m \in r} \sum_{s=1}^{S} \sum_{i_s \in V_c^{item}} x_{i_s,m} \leq n_c^{r,\max}$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.
- $n_c^{m,\min}$ and $n_c^{m,\max}$ specify the minimum and maximum required number of items from category level c in module m.
- $n_c^{r,\min}$ and $n_c^{r,\max}$ specify the minimum and maximum required number of items from category level c in pathway r.

See Also

`test_itemcat_con()`

Examples

```
data("mini_itempool")
# Example 1: same category ranges across all modules
# MST 1-3-3 design, 7 modules, 2 categories (MC, TEI).
test_mstATA <- mst_design(
  itempool = mini_itempool,
```

```
  design = "1-3-3",
  module_length = c(4,3,3,3,4,4,4)
)

test_itemcat_range_con(
  x = test_mstATA,
  attribute = "itemtype",
  cat_levels = c("MC","TEI"),
  min = matrix(c(2,1, rep(c(1,0),3), rep(c(1,0),3)), ncol = 2, byrow = TRUE),
  max = matrix(c(3,3, rep(c(6,1),3), rep(c(7,1),3)), ncol = 2, byrow = TRUE),
  target = NULL,
  deviation = NULL
)


# Example 2: different category ranges for each module
test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  module_length = c(4,3,3,3,4,4,4)
)

test_itemcat_range_con(
  x = test_mstATA,
  attribute = "itemtype",
  cat_levels = c("MC","TEI"),
  min = matrix(c(2,1, 1,0, 2,0, 1,0, 2,0, 3,1, 2,0),
               ncol = 2, byrow = TRUE),
  max = matrix(c(3,3, 2,2, 3,2, 3,1, 4,2, 4,2, 4,2),
               ncol = 2, byrow = TRUE),
  which_module = 1:7
)


# Example 3: same category range per pathway (target plus or minus deviation)
test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  module_length = c(4,3,3,3,4,4,4),
  exclude_pathways = c("1-1-3","1-3-1")
)

test_itemcat_range_con(
  x = test_mstATA,
  attribute = "itemtype",
  cat_levels = c("MC","TEI"),
  target  = c(9,2),
  deviation = 1,
  which_pathway = 1:7
)
```

`test_itemcount_con` *Generate Module- or Pathway-Level Constraints for the Number of Selected Items*

Description

This function generates linear constraints controlling the number of selected items in each module or pathway of a MST panel.

Depending on the MST design provided in `mst_design()`, the constraints enforce either:

- **Module-level item counts** - each module is treated as an independent test form
- **Pathway-level item counts** - each pathway is treated as an independent test form

The total number of constraints depends on the MST design and whether stage-level bounds are used:

- **Module-level constraints:** The number of constraints is **(number of modules)**
- **Pathway-level constraints:** The number of constraints is **(number of pathways) + (number of modules with a min bound) + (number of modules with a max bound) + (number of equality constraints enforcing equal module lengths within each stage)**

Usage

```
test_itemcount_con(x, stage_length_bound = NULL)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`stage_length_bound`	Optional `data.frame` with columns: `stage`, `min`, `max`, specifying minimum and maximum allowable numbers of items in each stage. If `NULL`, the default rule is: **at least one item per stage** (no empty modules), and modules within each stage must have equal lengths.

Details

1. Specification

The constraint enforces:

the test must include the exact number of *items*..

Key characteristics:

- The attribute type is *categorical*: each item has a unique ID.
- The attribute is defined at *item level* in the item pool.
- The constraint can be enforced at either **"Module-level"** or **"Pathway-level"**. It depends on whether `module_length` or `pathway_length` was specified in `mst_design()`.

2. Module-level constraints

These are generated when `module_length` was provided in `mst_design()`.

- The required number of items per module is given by `x$ModuleIndex$ModuleLength`.
- One constraint is created per module.
- Each module must contain *exactly* the specified number of items.

3. Pathway-level constraints

These are generated when `pathway_length` was provided in `mst_design()`.

- The required number of items per pathway is given by `x$PathwayIndex$PathwayLength`.
- Optional stage-level constraints may enforce:
  - Minimum items per stage
  - Maximum items per stage
- If `stage_length_bound = NULL`
  - Each stage must contain at least one item
  - All modules within a stage must have equal length

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

1. Module-level constraint

$$\sum_{s=1}^{S}\sum_{i_s=1}^{I_s} x_{i_s,m} = n_m$$

2. Pathway-level constraint

$$\sum_{m\in r}\sum_{s=1}^{S}\sum_{i_s=1}^{I_s} x_{i_s,m} = n_r$$

Stage-level minimum:

$$\sum_{s=1}^{S}\sum_{i_s=1}^{I_s} x_{i_s,m} \geq n_t^{\min}$$

for all modules m in stage t.

Stage-level maximum:

$$\sum_{s=1}^{S}\sum_{i_s=1}^{I_s} x_{i_s,m} \leq n_t^{\max}$$

for all modules m in stage t.

Equal-length constraint for modules (m, m') in the same stage:

$$\sum_{s=1}^{S}\sum_{i_s=1}^{I_s} x_{i_s,m} = \sum_{s=1}^{S}\sum_{i_s=1}^{I_s} x_{i_s,m'}$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.
- n_m specify the required number of items in module m.
- n_r specify the required number of items in pathway r.
- $n_t^{\min}$ specify the minimum number of items in stage t.
- $n_t^{\max}$ specify the maximum number of items in stage t.
- m, and m' are a pair of modules in stage t.

See Also

`mst_structure_con()`

Examples

```
data("mini_itempool")
# Example 1: module-level test length
# Routing module: 3 items; Stage 2 modules: 4 items each
test_mstATA <- mst_design(
  itempool = mini_itempool,module_length = c(3,4,4,4),
  design = "1-3"
)
test_itemcount_con(test_mstATA)

# Example 2: pathway-level test length: each pathway contains 7 items,
test_mstATA <- mst_design(
  itempool = mini_itempool,pathway_length = 7,
  design = "1-3"
)
# It will also constrain that each module has at least 1 item and modules
# in the stage 2 have the same module length.
test_itemcount_con(test_mstATA)

# Example 3: pathway-level with stage-level max bound for Stage 1
test_itemcount_con(
  test_mstATA,
  stage_length_bound = data.frame(stage = 1, min = 1, max = 2)
)
```

---

test_itemquant_con *Generate Module/Pathway-Level Constraints on the Sum of Item Quantitative Attributes*

Description

This function generates linear constraints on the **sum of item-level quantitative attributes** (e.g., total time, word count, item difficulty) within specified module or pathway of a multistage test (MST).

Users may specify:

- minimum total item-level attribute values,
- maximum total item-level attribute values, or
- exact (equal-to) total item-level attribute values.

Constraints may apply at the:

- **"Module-level"** – each module is treated as a separate test form, or
- **"Pathway-level"** – each pathway is treated as a complete test form.

The total number of constraints depends on the application level:

- **Module-level:** when `which_module` is provided OR both `which_module` and `which_pathway` are `NULL`

  The number of constraints is **(number of modules specified)**

- **Pathway-level:** when `which_pathway` is provided and `which_module = NULL`

  The number of constraints is **(number of pathways specified)**

Usage

```
test_itemquant_con(
  x,
  attribute,
  operator,
  target_value,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` containing the **item quantitative attribute** (e.g., time, difficulty, word count).
`operator`	A character string specifying the constraint type. Must be one of `"<="`, `">="`, or `"="`. For two-sided (range) constraints, use `test_itemquant_range_con()`.
`target_value`	A numeric scalar or vector giving the target sum of item quantitative attributes for each module/pathway being constrained.
`which_module`	Optional integer vector of module indices to which the constraints apply.
`which_pathway`	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforced is:

The total sum of the item quantitative attribute must meet a minimum, exact, or maximum bound for specific module or pathway.

Key characteristics:

- The attribute type is *quantitative*.
- The attribute is defined at the *item level* in the item pool.
- The constraints are applied at either **"Module-level"** or **"Pathway-level"**.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let q_{i_s} be the quantitative attribute value of item i_s.

1. Module-level constraint (for module m)

$$\sum_{s=1}^{S} \sum_{i_s=1}^{I_s} q_{i_s} x_{i_s,m} \overset{\leq}{\geq} b_q^{item,m}$$

2. Pathway-level constraint (for pathway r)

$$\sum_{m \in r} \sum_{s=1}^{S} \sum_{i_s=1}^{I_s} q_{i_s} x_{i_s,m} \overset{\leq}{\geq} b_q^{item,r}$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.
- The stacked operator, $\overset{\leq}{\geq}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- b_q is the required bound for the sum of item-level quantitative attribute values in module m (minimum, maximum, or exact value).
- b_q is the required bound for the sum of item-level quantitative attribute values in pathway r (minimum, maximum, or exact value).

Examples

```
data("mini_itempool")

# Example 1: Upper bound on total difficulty per module
test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  module_length = c(4,2,2,2,2,2,2)
)

# Mean difficulty upper bound per module:
#   difficulty_mean * module_length = difficulty_sum_target
test_itemquant_con(
  x = test_mstATA,
  attribute = "difficulty",
  operator = "<=",
  target_value = c(0, -0.5, 0, 0.5, -1, 0, 1) * c(4,2,2,2,2,2,2),
  which_module = NULL
)


# Example 2: Difficulty boundaries for different pathways
# Assume three subpopulations: low, medium, high ability.
# Pathways targeting low ability: mean difficulty <= -0.44
# Pathways targeting high ability: mean difficulty >= 0.44

test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  exclude_pathways = c("1-1-3","1-3-1"),
  pathway_length = 8
)

# Low-difficulty pathways
test_itemquant_con(
  x = test_mstATA,
  attribute = "difficulty",
  operator = "<=",
  target_value = -0.44 * 8,
  which_pathway = c(1,2)
)

# High-difficulty pathways
test_itemquant_con(
  x = test_mstATA,
  attribute = "difficulty",
  operator = ">=",
  target_value = 0.44 * 8,
  which_pathway = c(6,7)
)
```

`test_itemquant_range_con` *Generate Module/Pathway-Level Range Constraints for the Sum of Item Quantitative Attributes*

Description

This function generates linear constraints on the **sum of item-level quantitative attributes** (e.g., total time, word count, difficulty) within specified module or pathway of a multistage test (MST).

It supports two specification formats:

- **Direct range constraints:** using `min` and/or `max`
- **Target plus or minus deviation constraints:** using `target` and `deviation`

Constraints may apply at the:

- **"Module-level"** – each module is treated as a separate test form, or
- **"Pathway-level"** – each pathway is treated as a complete test form.

The total number of constraints depends on the application level:

- **Module-level:** when `which_module` is provided, or both `which_module` and `which_pathway` are `NULL`

  Number of constraints = **(number of modules) x side**
- **Pathway-level:** when `which_pathway` is provided and `which_module` = `NULL`

  Number of constraints = **(number of pathways) x side**

Here, `side` = `2` for range constraints (lower + upper bound) and `side` = `1` when only one bound is active.

Usage

```
test_itemquant_range_con(
  x,
  attribute,
  min = NULL,
  max = NULL,
  target = NULL,
  deviation = NULL,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

x	An object of class "mstATA_design" created by mst_design().
attribute	A string giving the column name in x$ItemPool containing the quantitative attribute (e.g., time, difficulty, word count).
min	Optional numeric scalar or vector of lower bounds.
max	Optional numeric scalar or vector of upper bounds.
target	Optional numeric scalar or vector of target values.
deviation	Optional numeric scalar or vector of allowed deviations from the target.
which_module	Optional integer vector of module indices to which the constraints apply.
which_pathway	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Constraint Specification

This function enforces:

The total sum of item quantitative attributes must lie within specified ranges for specified module or pathway.

The key properties of this constraint are:

- The attribute type is *quantitative*.
- The attribute is defined at the *item level* in the item pool.
- Constraints may be applied at the module or pathway level.

2. Range specification: min, max, target, deviation

Each of the four range-specification arguments may be: NULL, scalar or vector

Interpretation:

- min: minimum **sum** of item quantitative values
- max: maximum **sum** of item quantitative values
- target: desired **sum** of item quantitative values
- deviation: allowable deviation

When only target and deviation are provided: min = target - deviation. max = target + deviation.

Value

A combined constraint object of class "mstATA_constraint", consisting of one or more quantitative constraints.

name A string indicating the specification name.

specification A data.frame summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let q_{i_s} be the quantitative attribute value of item i_s.

1. Module-level constraint (for module m)

$$b_q^{m,\min} \leq \sum_{s=1}^{S} \sum_{i_s=1}^{I_s} q_{i_s} x_{i_s,m} \leq b_q^{m,\max}$$

2. Pathway-level constraint (for pathway r)

$$b_q^{r,\min} \leq \sum_{m \in r} \sum_{s=1}^{S} \sum_{i_s=1}^{I_s} q_{i_s} x_{i_s,m} \leq b_q^{r,\max}$$

Here:

- $x_{i_s,m}$ is the binary decision variable indicating whether item i_s is selected into module m.
- $m \in r$ denote modules belonging to pathway r.
- $b_q^{m,\min}$ and $b_q^{m,\max}$ specify the lower and upper bounds for the sum of item quantitative attribute in module m
- $b_q^{r,\min}$ and $b_q^{r,\max}$ specify the lower and upper bounds for the sum of item quantitative attribute in pathway r

See Also

`test_itemquant_con()`

Examples

```
data("mini_itempool")

## Example 1: Difficulty range per module
test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
```

```
  module_length = c(4,3,3,3,4,4,4)
)
test_itemquant_range_con(
  x = test_mstATA,
  attribute = "difficulty",
  min = c(-0.2*4, -0.7*2, -0.2*2, 0.3*2, -1.2*2, -0.2*2, 0.8*2),
  max = c( 0.2*4, -0.3*2,  0.2*2, 0.7*2, -0.8*2,  0.2*2, 1.2*2),
  which_module = NULL
)


## Example 2: Difficulty range per pathway
test_mstATA <- mst_design(
  itempool = mini_itempool,
  design = "1-3-3",
  exclude_pathways = c("1-1-3", "1-3-1"),
  pathway_length = 8
)

test_itemquant_range_con(
  x = test_mstATA,
  attribute = "difficulty",
  target = c(-1.2,-0.7,-0.2,0,0.2,0.7,1.2) * 8,
  deviation = 0.1 * 8,
  which_pathway = 1:7
)
```

`test_rdp_con` *Routing Decision Point Information Balance Constraint*

Description

Enforces similarity of test information at routing decision points (RDPs) between adjacent modules in the *next stage* of a multistage test (MST). Routing is assumed to occur **after examinees complete** `which_stage`, and constraints are imposed on modules at stage `which_stage + 1`.

Usage

```
test_rdp_con(x, rdp, which_stage, info_tol = 0.5)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`rdp`	A numeric vector of routing decision points (theta values). The length of `rdp` must equal the number of modules in stage `which_stage + 1` minus one. Each element defines the routing point between two adjacent modules in the next stage.

`which_stage` A single integer indicating the stage *after which* routing occurs. Routing constraints are applied to modules at stage `which_stage + 1`. Routing cannot be defined after the final stage.

`info_tol` A single positive numeric value specifying the maximum allowable difference in information between adjacent modules at each routing decision point.

Details

For each routing decision point θ_{rdp}, this function constrains the difference in test information functions (TIFs) between adjacent next-stage modules to be within a specified tolerance:

$$|I_{m_j}(\theta_r) - I_{m_{j+1}}(\theta_r)| \leq \tau$$

where τ is the information tolerance.

Adjacency is defined **by ordering of modules within the same stage**. This ensures correct behavior even when some pathways are excluded from the design.

The constraint enforces:

The test information at the routing decision point between adjacent modules in the next stage is similar within a specified tolerance.

Key characteristics:

- The attribute type is *quantitative*.
- The attribute is defined at the *item level* in the item pool.
- The constraints are applied at the **"Module-level"**.

This function requires that item-level information functions `iif(theta = theta_point)` already exist in the item pool. These attributes can be prepared in advance using `compute_iif`.

The constraint is implemented as a pair of linear inequalities for each routing decision point, resulting in two constraint rows per routing decision point.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index. Let $x_{i_s m}$ be a binary decision variable indicating whether item i_s from stimulus s is selected into module m.

Let θ_{rdp} denote a routing decision point (RDP) at which examinees are routed to different modules in the next stage of the test. The item information function of item i_s at ability level θ_r is denoted by I_{i_s}, and the corresponding module-level test information function is given by

$$I_m(\theta_{rdp}) = \sum_{s=1}^{S} \sum_{i_s=1}^{I_s} I_{i_s}(\theta_{rdp})\, x_{i_s m}.$$

Consider routing after stage k. Let $\mathcal{M}$ denote the ordered set of modules at stage $k+1$. For each pair of adjacent modules $(m_j, m_{j+1}$ in $\mathcal{M}$, a routing decision point θ_{r_j} is specified. The routing information balance constraint requires that the difference in test information between adjacent modules at the routing decision point be bounded by a user-specified tolerance $\delta > 0$:

$$|I_{m_j}(\theta_{r_j}) - I_{m_{j+1}}(\theta_{r_j})| \leq \delta.$$

Each absolute-value constraint is implemented as a pair of linear inequalities, resulting in two constraint rows per routing decision point.

See Also

compute_iif()

Examples

```
data("mini_itempool")
mini_itempool[,paste0("iif(theta=",c(-0.5,0.5),")")]<-compute_iif(mini_itempool,
item_par_cols = list("3PL"=c("discrimination","difficulty","guessing")),
                                                theta = c(-0.5,0.5),model_col = "model")
test_mstATA <- mst_design(
  itempool = mini_itempool,module_length = c(3,4,4,4),
  design = "1-3"
)
test_rdp_con(x = test_mstATA,rdp = c(-0.5, 0.5),
             which_stage = 1,info_tol = 0.4)
```

`test_stimcat_con` *Generate Module/Pathway-Level Constraints for the Min/Exact/Max Number of Stimuli from Specific Categorical Levels*

Description

This function generates linear constraints on the number of *stimuli* belonging to specified categorical levels (e.g., passage type, theme, genre) in a multistage test (MST).

Constraints may be applied at:

- **"Module-level"** – each module is treated as an independent form.
- **"Pathway-level"** – each pathway is treated as an independent form.

The application level is determined by which of `which_module` or `which_pathway` is supplied.

The total number of constraints depends on the application level and on the number of categorical levels included in `cat_levels`:

- **Module-level:** when `which_module` is provided OR both `which_module` and `which_pathway` are `NULL`

  The number of constraints is **(number of category levels) x (number of modules specified)**
- **Pathway-level:** when `which_pathway` is provided and `which_module = NULL`

  The number of constraints is **(number of category levels) x (number of pathways specified)**

Users may constrain only a subset of modules or pathways using `which_module` or `which_pathway`, and may restrict the constraints to a subset of stimulus categories using `cat_levels`.

Usage

```
test_stimcat_con(
  x,
  attribute,
  cat_levels,
  operator,
  target_num,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that represents the **stimulus-level categorical attribute**.
`cat_levels`	Character string or vector of category levels to constrain.
`operator`	A character string indicating the type of constraint. Must be one of `"<="`, `"="`, or `">="`.

`target_num`	Target number of stimuli per category per module/pathway. Can be a scalar, vector, or matrix (see Details).
`which_module`	Optional integer vector of module indices to which the constraints apply.
`which_pathway`	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforces:

The test must meet a minimum, exact, or maximum number of stimuli belonging to each specified categorical level.

Key characteristics:

- The attribute type is *categorical*.
- The attribute is defined at the *stimulus level* in the item pool.
- The constraints are applied at either **"Module-level"** or **"Pathway-level"**.

2. Pivot-Item Method

MST assembly uses the **pivot-item formulation** (van der Linden, 2005):

- Each stimulus has one designated pivot item i_s^*.
- A stimulus is considered selected if and only if its pivot item is selected.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let V_c^{stim} denote the set of stimuli that belong to category c.

1. Module-level constraint (for module m)

$$\sum_{s \in V_c^{stim}} x_{i_s^*, m} \overset{\leq}{\geq} n_c^{stim,m}$$

2. Pathway-level constraint (for pathway r)

$$\sum_{m \in r} \sum_{s \in V_c^{stim}} x_{i_s^*, m} \overset{\leq}{\geq} n_c^{stim,r}$$

Here:

- $x_{i_s^*, m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus is selected in that module.
- $m \in r$ denote modules belonging to pathway r.
- The stacked operator, $\overset{\leq}{\geq}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- $n_c^{stim,m}$ specify the required number of stimuli from category level c in module m.
- $n_c^{stim,r}$ specify the required number of stimuli from category level c in pathway r.

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool   = reading_itempool,
  stimulus   = "stimulus",
  pivot_item = "pivot_item"
)

# Example 1:
# MST 1-3-3 design, 7 modules, 2 stimulus categories (history, social studies).
# Stage 1: less than 1 history passage.
# Stage 2 & 3: less than 1 history and less than 1 social studies passage in each module.

test_mstATA <- mst_design(
  itempool = reading_itempool,
  design   = "1-3-3",
  module_length = c(14,12,12,12,12,12,12),
  pivot_stim_map = pivot_stim_map
```

```
)

test_stimcat_con(
  x = test_mstATA,
  attribute = "stimulus_type",
  cat_levels = c("history","social studies"),
  operator = "<=",
  target_num = matrix(
    c(1,0,
      rep(c(1,1), 3),
      rep(c(1,1), 3)),
    nrow = 7, ncol = 2, byrow = TRUE
  )
)


# Example 2:
# Pathway REE requires between 1 and 3 history passages.
con1<-test_stimcat_con(
    x = test_mstATA,
    attribute = "stimulus_type",
    cat_levels = "history",
    operator = ">=",
    target_num = 1,
    which_pathway = 1
  )
con2<-test_stimcat_con(
    x = test_mstATA,
    attribute = "stimulus_type",
    cat_levels = "history",
    operator = "<=",
    target_num = 3,
    which_pathway = 1
  )
```

`test_stimcount_con` *Generate Module/Pathway-Level Constraints for the Number of Selected Stimuli*

Description

This function generates linear constraints on the *number of stimuli* selected in a multistage test (MST).

Constraints may be applied at:

- **"Module-level"** – each module is treated as an independent form.
- **"Pathway-level"** – each pathway is treated as an independent form.

The application level is determined by which of `which_module` or `which_pathway` is supplied.

The total number of constraints is:

- **Module-level:** when `which_module` is provided, or when both `which_module` and `which_pathway` are `NULL`
  The number of constraints is **(number of modules specified)**
- **Pathway-level:** when `which_pathway` is provided and `which_module` = `NULL`
  The number of constraints is **(number of pathways specified)**

Usage

```
test_stimcount_con(
  x,
  operator,
  target_num,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`operator`	A character string indicating the type of constraint. Must be one of `"<="`, `"="`, or `">="`.
`target_num`	The number of stimuli in specific module/pathway (`which_module`, `which_pathway`).
`which_module`	Optional integer vector of module indices to which the constraints apply.
`which_pathway`	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforces:

The test must meet a minimum, exact, or maximum number of stimuli.

Key characteristics:

- The attribute type is *categorical* (each stimulus has a unique ID).
- The attribute is defined at the *stimulus level* in the item pool.
- The constraints are applied at either **"Module-level"** or **"Pathway-level"**.

2. Pivot-Item Method

MST assembly uses the **pivot-item formulation** (van der Linden, 2005):

- Each stimulus has one designated pivot item i_s^*.
- A stimulus is considered selected if and only if its pivot item is selected.

Value

An object of S3 class "mstATA_constraint" with named elements:

name A character vector indicating the specifications in each row of A_binary

specification A data.frame summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of A_binary.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

1. Module-level constraint (for module m)

$$\sum_{s=1}^{S-1} x_{i_s^*,m} \begin{matrix}\leq \\ \geq \\ = \end{matrix} n_m$$

2. Pathway-level constraint (for pathway r)

$$\sum_{m \in r} \sum_{s=1}^{S-1} x_{i_s^*,m} \begin{matrix}\leq \\ \geq \\ = \end{matrix} n_r$$

Here:

- $x_{i_s^*,m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus is selected in that module.
- $m \in r$ denote modules belonging to pathway r.
- The stacked operator, $\begin{matrix}\leq \\ \geq \\ = \end{matrix}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- n_m specify the required number of stimuli in module m.
- n_r specify the required number of stimuli in pathway r.

References

van der Linden, W. J. (2005). *Linear Models for Optimal Test Design*. Springer. doi:10.1007/0387290540.

Examples

```
data("reading_itempool")

pivot_stim_map <- create_pivot_stimulus_map(
  itempool = reading_itempool,
  stimulus = "stimulus",
  pivot_item = "pivot_item"
)

test_mstATA <- mst_design(
  itempool = reading_itempool,
  design = "1-3-3",
  module_length = c(14,12,12,12,12,12,12),
  pivot_stim_map = pivot_stim_map
)

# Example 1:
# In MST 1-3-3, the routing module must contain between 1 and 2 stimuli.
con1<-test_stimcount_con(
    x = test_mstATA,
    operator = ">=",
    target_num = 1,
    which_module = 1
  )
con2<-test_stimcount_con(
    x = test_mstATA,
    operator = "<=",
    target_num = 2,
    which_module = 1
  )


# Example 2:
# The REE pathway must contain exactly 2 stimuli.
test_stimcount_con(
  x = test_mstATA,
  operator = "=",
  target_num = 2,
  which_pathway = 1
)
```

`test_stimquant_con` *Generate Module/Pathway-Level Constraints for the Sum of Stimulus Quantitative Attributes*

Description

This function generates linear constraints on the **sum of stimulus-level quantitative attribute** (e.g., total word count, total reading time, total linguistic complexity) in a multistage test (MST).

Users may specify:

- minimum total stimulus-level quantitative attribute values,
- maximum total stimulus-level quantitative attribute values, or
- exact (equal-to) total stimulus-level attribute values.

Constraints may be applied at:

- **"Module-level"** – each module is treated as an independent form.
- **"Pathway-level"** – each pathway is treated as an independent form.

The total number of constraints depends on the application level:

- **Module-level:**when `which_module` is provided OR both `which_module` and `which_pathway` are `NULL`
  The number of constraints is **(number of modules specified)**
- **Pathway-level:**when `which_pathway` is provided and `which_module` = `NULL`
  The number of constraints is **(number of pathways specified)**

Usage

```
test_stimquant_con(
  x,
  attribute,
  operator,
  target_value,
  which_module = NULL,
  which_pathway = NULL
)
```

Arguments

`x`	An object of class `"mstATA_design"` created by `mst_design()`.
`attribute`	A string giving the column name in `x$ItemPool` that represents the **stimulus-level quantitative attribute**.
`operator`	A character string indicating the type of constraint. Must be one of `"<="`, `"="`, or `">="`.
`target_value`	A numeric scalar or vector specifying the required total quantitative attribute value.
`which_module`	Optional integer vector of module indices to which the constraints apply.
`which_pathway`	Optional integer vector of pathway indices to which the constraints apply.

Details

1. Specification

The constraint enforces:

The total value of quantitative attribute across selected stimuli must meet a minimum, exact, or maximum requirement.

Example:In a stimulus-based assessment, total words in each module must not exceed 300.

Key characteristics:

- The attribute type is *quantitative*.
- The attribute is defined at the *stimulus level* in the item pool.
- The constraints are applied at either **"Module-level"** or **"Pathway-level"**.

2. Pivot-Item Method

MST assembly uses the **pivot-item formulation** (van der Linden, 2005):

- Each stimulus has one designated pivot item i_s^*.
- A stimulus is considered selected if and only if its pivot item is selected.

Value

An object of S3 class `"mstATA_constraint"` with named elements:

name A character vector indicating the specifications in each row of `A_binary`

specification A `data.frame` summarizing the constraint specification, including the requirement name, attribute, constraint type, application level, operator, and the number of constraint rows generated.

A_binary A sparse binary matrix representing the linear constraint coefficients.

A_real NULL for 'mstATA_constraint' object

operators A character vector of constraint operators, one per row of `A_binary`.

d A numeric vector of right-hand-side values for the constraints.

C_binary NULL for 'mstATA_constraint' object

C_real NULL for 'mstATA_constraint' object

sense NULL for 'mstATA_constraint' object

Mathematical Formulation

Suppose the item pool contains (S - 1) stimulus-based item sets, indexed by $s = 1, \ldots, S-1$. Each stimulus has a designated pivot item, indexed by i_s^*. In addition, the pool contains a set of discrete (non-stimulus-based) items, which are represented by a dummy stimulus $s = S$ to allow a unified indexing scheme. Items belonging to stimulus s are indexed as $i_s = 1, 2, \ldots, I_s$.

Suppose there are M modules in an MST panel. Let $m = 1, \ldots, M$ denote the module index.

Let q_s be the quantitative attribute value of stimulus s.

1. Module-level constraint (for module m)

$$\sum_{s=1}^{S-1} q_s x_{i_s^*,m} \overset{\leq}{\underset{=}{\geq}} b_q^{stim,m}$$

2. Pathway-level constraint (for pathway r)

$$\sum_{m \in r} \sum_{s=1}^{S-1} q_s x_{i_s^*,m} \overset{\leq}{\underset{=}{\geq}} b_q^{stim,r}$$

Here:

- $x_{i_s^*,m}$ indicates whether the pivot item for stimulus s is selected into module m, thereby indicating whether the stimulus is selected in that module.
- $m \in r$ denote modules belonging to pathway r.
- The stacked operator, $\overset{\leq}{\underset{=}{\geq}}$ denotes that the specification may take the form of an upper bound, lower bound, or exact value.
- b_q is the required bound for the sum of stimulus-level quantitative attribute values in module m.
- b_q is the required bound for the sum of stimulus-level quantitative attribute values in pathway r.

Examples

```
data("reading_itempool")
pivot_stim_map <- create_pivot_stimulus_map(reading_itempool,
                                            item_id_col="item_id",
                                            stimulus = "stimulus",
                                             pivot_item = "pivot_item")
test_mstATA <- mst_design(
  itempool = reading_itempool,
  design = "1-3-3",
  module_length = c(14,12,12,12,12,12,12),
  pivot_stim_map = pivot_stim_map
)

# Example 1:
# The total words in each module must be less than 300.
test_stimquant_con(
  x = test_mstATA,
  attribute = "stimulus_words",
  operator = "<=",
  target_value = 300,
  which_module = NULL
)


# Example 2:
# In the REE pathway, total stimulus words must be between 50 and 200.
con1<-test_stimquant_con(
```

```
    x = test_mstATA,
    attribute = "stimulus_words",
    operator = ">=",
    target_value = 50,
    which_pathway = 1
  )
con2<-test_stimquant_con(
    x = test_mstATA,
    attribute = "stimulus_words",
    operator = "<=",
    target_value = 200,
    which_pathway = 1
  )
```

`unary_minimax_panel` *Precomputed MST Panel: Unary_Minimax Formulation*

Description

A precomputed multistage testing (MST) panel assembled using the minimax objective formulation (one_dev) described in the *Formulation for Multiple Objectives* vignette.

Usage

```
unary_minimax_panel
```

Format

An object of class `mstATA_panel`.

Details

This object is included to avoid long solver runtimes during vignette building and package checking.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

`weighted_sum_obj` *Minimize or Maximize a Weighted Sum of Multiple Objective Functions*

Description

This function compiles a **weighted-sum objective** when multiple objective terms are present. Each objective term contributes a linear score, and the solver optimizes a weighted linear combination of these scores.

Under this strategy, multiple linear objective terms are aggregated into a single objective function by forming a weighted linear combination.

Usage

```
weighted_sum_obj(x, multiple_terms, strategy_args = list())
```

Arguments

`x`	An object of class "mstATA_design" created by mst_design().
`multiple_terms`	A list containing **two or more** objective components. Each element must be either: **An objective term** created using objective_term(). **A compiled objective** produced by a multi-objective formulation such as maximin_obj(), capped_maximin_obj(), minimax_obj(), or goal_programming_obj(). Each element may represent either a relative or an absolute objective. The weighted aggregation respects the optimization sense (maximize or minimize) specified within each objective term or compiled objective.
`strategy_args`	A named list of strategy-specific arguments used by the **weighted_sum** aggregation. Supported fields: • `weights` – A positive numeric vector of length `length(multiple_term)` specifying the relative importance of each objective term. Defaults to a vector of ones.

Details

1. Overview

There are at least two objective term.

Each objective term represents a linear score of the form $a^\top x$, constructed by objective_term().

Key characteristics for each objective term:

(1) Categorical or Quantitative

- **maximize or minimize the number of items belonging to a specific categorical level**, or

- **maximize or minimize the sum of quantitative item-level attribute values**, such as difficulty, discrimination, or item information function values.

(2) Module-, Pathway- or Panel-level

- **"Module-level"** – only items in a specified module contribute to the objective;
- **"Pathway-level"** – items in all modules belonging to one pathway contribute; and
- **"Panel-level"** – all item selections in the panel contribute.

(3) Relative or Absolute Objective

- **Relative objective**: maximize/minimize $a^\top x$.
- **Absolute deviation objective**: minimize the deviation from a specified target: $|a^\top x - g|$ where g is the target value.

To combine these objectives, the function introduces an auxiliary continuous variable y_k for each term and optimizes a weighted sum:

$$\min \sum_{k=1}^{K} w_k y_k$$

or

$$\max \sum_{k=1}^{K} w_k y_k$$

where $w_k > 0$ are user-specified or default weights.

2. Handling Mixed Optimization Directions

Objective terms may have different optimization directions:

- some terms are to be minimized, and
- others are to be maximized.

To maintain a single optimization sense, the compiled objective always uses a **minimization** form internally when mixed directions are present.

For any term that is originally a maximization objective, its score is multiplied by -1. For example, if:

- y_1 and y_2 are to be minimized, and
- y_3 is to be maximized,

then the combined objective becomes:

$$\min(w_1 y_1 + w_2 y_2 - w_3 y_3)$$

This transformation preserves the intended optimization direction for each objective term while yielding a single linear objective function.

3.Practical Considerations for Choosing Weights

As discussed in van der Linden (2005, pp. 68-69), selecting weights in a multiobjective test-assembly problem can be challenging, particularly when the objective terms are measured on different scales (e.g., test information, content counts, exposure-related quantities).

Two key issues arise:

- When objectives are on different scales, weights are difficult to interpret and compare meaningfully.
- When several weights are chosen to be close in magnitude, the optimal solution may become unstable or unpredictable due to strong trade-offs between competing attributes.

A partial remedy suggested by van der Linden is to first solve the assembly problem separately for each objective term. These single-objective runs yield benchmark values corresponding to solutions in which the entire weight is placed on one objective and zero weight on all others.

The final weights in a weighted-sum formulation can then be selected by compromising between these benchmark values, providing a more informed and transparent basis for multiobjective trade-offs.

This strategy is especially useful during test design and exploratory phases, where understanding the attainable extremes of each objective helps guide principled weight selection.

How to use:

- Define individual objective terms using objective_term(). Objective terms may be entirely relative, entirely absolute, or a mixture of both.
- Combine them directly using weighted_sum_obj().
- Alternatively, first construct compiled objectives using maximin_obj(), capped_maximin_obj(), minimax_obj(), or goal_programming_obj(), and then combine multiple compiled objectives using weighted_sum_obj().

Value

A list of class "compiled_objective".

name A character vector indicating the specifications in each row of "A_binary"

specification A `data.frame` including "Requirement","Attribute","Type","Application Level","Operator","Num of Constraints"

A_binary Sparse matrix of coefficients for binary decision variables.

A_real Sparse matrix of coefficients for real decision variables.

operators A character vector of constraint operators, one per row of "A_binary".

d A numeric vector of right-hand-side values for the constraints.

C_binary Penalty vector for binary variables (if any)

C_real Penalty vector for real variables (if any)

sense "min" or "max"

decisionvar_name_new Character vector indicating the names of new decision variables, same length as "ncol(A_real)".

decisionvar_type_new A character vector of "C" (continuous), same length as "ncol(A_real)".

notes List of metadata.

Mathematical Formulation

For each objective term $k = 1, \ldots, K$:

$$y_k = a_k^\top x$$

where:

For relative objectives:

- a_k is the coefficient vector defined by the attribute, level, and application scope of the objective term;
- x is the vector of binary item-module-panel decision variables;
- the objective value is $y_k = a_k^\top x$.

For absolute (goal-based) objectives:

- a_k is the coefficient vector defined by the attribute, level, and application scope of the objective term;
- g_k is the target value specified by the test developer;
- the objective minimizes deviation from the target: $|a_k^\top x - g_k|$;
- auxiliary deviation variables are introduced to linearize the absolute value.

The weighted-sum objective is then:

$$\min \sum_{k=1}^{K} w_k \tilde{y}_k$$

where: $\tilde{y}$ if term k is a minimization objective. $\tilde{y}$ if term k is a maximization objective.

References

van der Linden, W. J. (2005). *Linear models for optimal test design* (pp. 68–69). doi:10.1007/0387290540

Examples

```
data("mini_itempool")
test_mstATA <- mst_design(
  itempool        = mini_itempool,
  design          = "1-3-3",
  exclude_pathway = c("1-1-3", "1-3-1"),
  pathway_length  = 8
)

# Example: maximize test information at theta = -1, 0, and 1
# in the routing module, with targets at -1 and 1

obj1 <- objective_term(
  x             = test_mstATA,
```

```
  attribute     = "iif(theta=-1)",
  applied_level = "Module-level",
  which_module  = 1,
  sense         = "min",
  goal          = 12
)

obj2 <- objective_term(
  x             = test_mstATA,
  attribute     = "iif(theta=0)",
  applied_level = "Module-level",
  which_module  = 1,
  sense         = "max"
)

obj3 <- objective_term(
  x             = test_mstATA,
  attribute     = "iif(theta=1)",
  applied_level = "Module-level",
  which_module  = 1,
  sense         = "min",
  goal          = 12
)


weighted_sum_obj(x = test_mstATA, multiple_terms = list(obj1,obj2,obj3))
```

`weighted_sum_panel` *Precomputed MST Panel: Weighted_Sum Formulation*

Description

A precomputed multistage testing (MST) panel assembled using the weighted_sum objective formulation described in the *Formulation for Multiple Objectives* vignette.

Usage

```
weighted_sum_panel
```

Format

An object of class `mstATA_panel`.

Details

This object is included to avoid long solver runtimes during vignette building and package checking.

Source

Generated using `mstATA::assembled_panels()` and `mstATA::solve_model()`with the HiGHS solver.

Index