

# Package ‘MultiObjMatch’

May 7, 2026

**Type** Package

**Title** Multi-Objective Matching Algorithm

**Version** 1.0.0

**Description** Matching algorithm based on network-flow structure.

Users are able to modify the emphasis on three different optimization goals: two different distance measures and the number of treated units left unmatched. The method is proposed by Pimentel and Kelz (2019) <[doi:10.1080/01621459.2020.1720693](https://doi.org/10.1080/01621459.2020.1720693)>.

The 'rrelaxiv' package, which provides an alternative solver for the underlying network flow problems, carries an academic license and is not available on CRAN, but may be downloaded from Github at <<https://github.com/josherrickson/rrelaxiv/>>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Imports** cobalt, dplyr, optmatch, matchMulti, fields, plyr, RCurl, gtools, rcbalance, MASS, stats, methods, ggplot2, utils, rlang, rlemon

**RoxygenNote** 7.1.2

**Suggests** testthat (>= 3.0.0), rrelaxiv

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Shichao Han [cre, aut],  
Samuel D. Pimentel [aut]

**Maintainer** Shichao Han <[schan21@berkeley.edu](mailto:schan21@berkeley.edu)>

**Repository** CRAN

**Date/Publication** 2024-07-04 23:10:05 UTC

## Contents

addBalance . . . . .	3
----------------------	---

addExclusion	3
balanceCosts	4
build.dist.struct	4
build.dist.struct_user	5
callrelax	6
check_representative	6
combine_dist	7
combine_match_result	7
compare_matching	8
compare_tables	9
convert_index	10
convert_names	10
costSkeleton	11
data_precheck	11
descr.stats_general	12
distanceFunctionHelper	12
dist_bal_match	13
dummy	16
edgelist2ISM	17
excludeCosts	17
extractEdges	18
extractSupply	18
filter_match_result	19
flattenSkeleton	19
generateRhoObj	20
generate_rhos	20
getExactOn	21
getPropensityScore	21
get_balance_table	22
get_five_index	23
get_pairdist_balance_graph	24
get_pairdist_graph	25
get_rho_obj	26
get_tv_graph	26
get_unmatched	27
makeInfinitySparseMatrix	28
makeSparse	29
matched_data	29
matched_index	30
matrix2cost	30
matrix2edgelist	31
meldMask	31
netFlowMatch	32
obj.to.match	32
pairCosts	33
rho_proposition	33
solveP	34
solveP1	34

<i>addBalance</i>	3
summary.multiObjMatch . . . . .	35
two_dist_match . . . . .	36
visualize . . . . .	40

**Index** **42**

*addBalance*                      *Add fine balance edges*

**Description**

Add fine balance edges

**Usage**

`addBalance(net, treatedVals, controlVals, replaceExisting = TRUE)`

**Arguments**

<code>net</code>	the network object created for the network flow problem
<code>treatedVals</code>	the balance value for treated nodes
<code>controlVals</code>	the balance value for control nodes
<code>replaceExisting</code>	(optional) whether or not to replace the existing net; TRUE by default

**Value**

the network structure with balance edges added

*addExclusion*                      *Add exclusion edges*

**Description**

Add exclusion edges

**Usage**

`addExclusion(net, remove = FALSE)`

**Arguments**

<code>net</code>	the input network structure
<code>remove</code>	(optional) whether to exclude edges; FALSE by default

**Value**

the network structure with exclusion edges added to allow for tradeoff for the exclusion cost

balanceCosts                    *Create a skeleton representation of the balance edge costs*

---

### Description

Create a skeleton representation of the balance edge costs associated with pairings for a given distance and network

### Usage

```
balanceCosts(net, balance.penalty = 1)
```

### Arguments

net                    the network structure  
balance.penalty        (optional) the numeric value for balance; 1 by default

### Value

the skeleton with balance edge cost

---

build.dist.struct        *An internal helper function that generates the data abstraction for the edge weights of the main network structure.*

---

### Description

An internal helper function that generates the data abstraction for the edge weights of the main network structure.

### Usage

```
build.dist.struct(  
  z,  
  X,  
  distMat,  
  exact = NULL,  
  dist.type = "Mahalanobis",  
  calip.option = "propensity",  
  calip.cov = NULL,  
  caliper = 0.2,  
  verbose = FALSE  
)
```

**Arguments**

z	a vector of treatment and control indicators, 1 for treatment and 0 for control.
X	a data frame or a numeric or logical matrix containing covariate information for treated and control units. Its row count must be equal to the length of z.
distMat	a matrix of pair-wise distance specified by the user
exact	an optional vector of the same length as z. If this argument is specified, treated units will only be allowed to match to control units that have equal values in the corresponding indices of the exact vector. For example, to match patients within hospitals only, one could set exact equal to a vector of hospital IDs for each patient.
dist.type	one of ('propensity', 'user', 'none'). If 'propensity' is specified (the default option), the function estimates a propensity score via logistic regression of z on X and imposes a propensity score caliper. If 'user' is specified, the user must provide a vector of values on which a caliper will be enforced using the calip.cov argument. If 'none' is specified no caliper is used.
calip.option	a character indicating the type of caliper used
calip.cov	see calip.option.
caliper	a numeric value that gives the size of the caliper when the user specifies the calip.option argument as 'propensity' or 'calip.cov'.
verbose	a boolean value whether to print(cat) debug information. Default: FALSE

**Value**

a distance structure used for constructing the main network flow problem

---

build.dist.struct\_user

*An internal helper function that generates the data abstraction for the edge weights of the main network structure using the distance matrix passed by the user.*

---

**Description**

An internal helper function that generates the data abstraction for the edge weights of the main network structure using the distance matrix passed by the user.

**Usage**

```
build.dist.struct_user(z, distMat, verbose = FALSE)
```

**Arguments**

z	a vector indicating whether each unit is in treatment or control group
distMat	a matrix of pair-wise distance
verbose	a boolean value whether to print(cat) debug information. Default: FALSE

**Value**

a distance structure used for constructing the main network flow problem

---

callrelax	<i>Call relax on the network</i>
-----------	----------------------------------

---

**Description**

this function is copied from the rbalance package

**Usage**

```
callrelax(net, solver = "rlemon")
```

**Arguments**

net	the network structure
solver	(optional) the solver; by default, "rlemon"

**Value**

list of the result from the call to relax solver

---

check_representative	<i>Check the representativeness of matched treated units</i>
----------------------	--

---

**Description**

Summary function to compare SMD of the key covariates in matched and the full set of treated units.

**Usage**

```
check_representative(matching_result, match_num = NULL)
```

**Arguments**

matching_result	the matching result returned by either dist_bal_match or two_dist_match.
match_num	(optional) Integer index of match that the user want to extract paired observations from. NULL by default, which will generate a table for all the matches.

**Value**

a summary table of SMDs of the key covariates between the whole treated units and the matched treated units.

---

combine_dist	<i>An internal helper function that combines two distance object</i>
--------------	--

---

**Description**

An internal helper function that combines two distance object

**Usage**

```
combine_dist(a, b)
```

**Arguments**

a	a distance structure object
b	a distance structure object

**Value**

a new distance structure object whose edge weights are the sum of the corresponding edge weights in a and b

---

combine_match_result	<i>Combine two matching result</i>
----------------------	------------------------------------

---

**Description**

Combine two matching result

**Usage**

```
combine_match_result(matching_result1, matching_result2)
```

**Arguments**

matching_result1	the first matching result object.
matching_result2	the second matching result object.

**Value**

a new matching result combining two objects. Note that the matching index for the second matching is the original name plus the maximum match index in the first matching object.

---

compare_matching	<i>Generate covariate balance in different matches</i>
------------------	--

---

## Description

This is a wrapper function for use in evaluating covariate balance across different matches. It only works for 'Basic' version of matching (using `dist_bal_match`).

## Usage

```
compare_matching(
  matching_result,
  cov_list = NULL,
  display_all = TRUE,
  stat = "mean.diff"
)
```

## Arguments

<code>matching_result</code>	an object returned by the main matching function <code>dist_bal_match</code>
<code>cov_list</code>	(optional) factor of names of covariates that we want to evaluate covariate balance on; default is <code>NULL</code> . When set to <code>NULL</code> , the program will compare the covariates that have been used to construct a propensity model.
<code>display_all</code>	(optional) boolean value of whether to display all the matches; default is <code>TRUE</code> , where matches at each quantile is displayed
<code>stat</code>	(optional) character of the name of the statistic used for measuring covariate balance; default is "mean.diff". This argument is the same as used in "cobalt" package, see: <a href="#">bal.tab</a>

## Value

a dataframe that shows covariate balance in different matches

## Examples

```
## Generate matches
data("lalonde", package="cobalt")
ps_cols <- c("age", "educ", "married", "nodegree", "race")
treat_val <- "treat"
response_val <- "re78"
pair_dist_val <- c("age", "married", "educ", "nodegree", "race")
my_bal_val <- c("race")
r1s <- c(0.01, 1, 2, 4, 4.4, 5.2, 5.4, 5.6, 5.8, 6)
r2s <- c(0.001)
match_result <- dist_bal_match(data=lalonde, treat_col= treat_val,
  marg_bal_col = my_bal_val, exclusion_penalty=r1s, balance_penalty=r2s,
```

```

dist_col = pair_dist_val,
propensity_col = ps_cols, max_iter=0)

## Generate table for comparing matches
compare_matching(match_result, display_all = TRUE)

```

---

compare_tables	<i>Summarize covariate balance table</i>
----------------	--

---

### Description

This function would take the result of `get_balance_table` function and combine the results in a single table. It only works for 'Basic' version of the matching.

### Usage

```
compare_tables(balance_table)
```

### Arguments

`balance_table` a named list, which is the result from the function `get_balance_table`

### Value

a dataframe with combined information

### Examples

```

## Generate matches
data("lalonge", package="cobalt")
ps_cols <- c("age", "educ", "married", "nodegree", "race")
treat_val <- "treat"
response_val <- "re78"
pair_dist_val <- c("age", "married", "educ", "nodegree", "race")
my_bal_val <- c("race")
r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
match_result <- dist_bal_match(data=lalonge, treat_col= treat_val,
marg_bal_col = my_bal_val, exclusion_penalty=r1s, balance_penalty=r2s,
dist_col = pair_dist_val,
propensity_col = ps_cols, max_iter=0)

## Generate summary table for comparing matches
compare_tables(get_balance_table(match_result))

```

---

convert_index	<i>An internal helper function that translates the matching index in the sorted data frame to the original dataframe's row index</i>
---------------	--

---

**Description**

An internal helper function that translates the matching index in the sorted data frame to the original dataframe's row index

**Usage**

```
convert_index(matching_result)
```

**Arguments**

matching\_result  
an object returned by the main matching function dist\_bal\_match

---

convert_names	<i>Internal helper function that converts axis name to internal variable name</i>
---------------	---

---

**Description**

Internal helper function that converts axis name to internal variable name

**Usage**

```
convert_names(x, y, z = NULL)
```

**Arguments**

x            the user input character for x-axis value  
y            the user input character for y-axis value  
z            the user input character for z-axis value

**Value**

a named list with variable names for visualization for internal use

---

costSkeleton	<i>Create cost skeleton</i>
--------------	-----------------------------

---

**Description**

Create a more user-friendly data structure to represent the edge costs in a network. Internally the network object used by the optimization routine represents all the edge costs in a single vector. The "skeleton" structure decomposes this vector into a list of components, each corresponding to a different role in the network: "pairings" are edges between treated and control, "exclusion" are direct links between treated units and a sink that allows them to be excluded, "balance" refers to edges that count marginal balance between groups, and "sink" indicates edges that connect control nodes to the sink. Skeletons are created so these various features can be combined (or switched on and off) easily into objective functions, and the interface to the main tradeoff function expects to see each function represented in skeleton format.

**Usage**

```
costSkeleton(net)
```

**Arguments**

net	the network structure
-----	-----------------------

**Value**

the skeleton

---

data_precheck	<i>Data precheck: Handle missing data(mean imputation) and remove redundant columns; it also adds an NA column for indicating whether it's missing</i>
---------------	--

---

**Description**

Data precheck: Handle missing data(mean imputation) and remove redundant columns; it also adds an NA column for indicating whether it's missing

**Usage**

```
data_precheck(X)
```

**Arguments**

X	a dataframe that the user initially inputs for matching - dataframe with covariates
---	---

**Value**

a dataframe with modified data if necessary

---

`descr.stats_general`    *Generate summary statistics for matches*

---

### Description

Generate summary statistics for matches

### Usage

```
descr.stats_general(matches, df, treatCol, b.vars, pair.vars, extra = FALSE)
```

### Arguments

<code>matches</code>	One matching result from the main matching function
<code>df</code>	the original data frame used for matching
<code>treatCol</code>	the character of the column name for treatment vector
<code>b.vars</code>	the vector of column names of covariates used for measuring balance
<code>pair.vars</code>	the vector of column names used for measuring pairwise distance
<code>extra</code>	the list of summary statistic; it must be the types that can be taken by cobalt

### Value

a named vector of summary statistic

---

`distanceFunctionHelper`  
*Helper function that change input distance matrix*

---

### Description

Helper function that change input distance matrix

### Usage

```
distanceFunctionHelper(z, distMat)
```

### Arguments

<code>z</code>	the treatment vector
<code>distMat</code>	the user input distance matrix

### Value

a distance matrix where (i,j) element is the distance between unit i and j in the same order as z

---

dist_bal_match	<i>Optimal tradeoffs among distance, exclusion and marginal imbalance</i>
----------------	---

---

### Description

Explores tradeoffs among three important objective functions in an optimal matching problem: the sum of covariate distances within matched pairs, the number of treated units included in the match, and the marginal imbalance on pre-specified covariates (in total variation distance).

### Usage

```
dist_bal_match(
  data,
  treat_col,
  marg_bal_col,
  exclusion_penalty = c(),
  balance_penalty = c(),
  dist_matrix = NULL,
  dist_col = NULL,
  exact_col = NULL,
  propensity_col = NULL,
  pscore_name = NULL,
  ignore_col = NULL,
  max_unmatched = 0.25,
  caliper_option = NULL,
  tol = 0.01,
  max_iter = 1,
  rho_max_factor = 10,
  max_pareto_search_iter = 5
)
```

### Arguments

data	data frame that contain columns indicating treatment, outcome and covariates.
treat_col	character of name of the column indicating treatment assignment.
marg_bal_col	character of column name of the variable on which to evaluate marginal balance.
exclusion_penalty	(optional) numeric vector of values of exclusion penalty. Default is c(), which would trigger the auto grid search.
balance_penalty	(optional) factor of values of marginal balance penalty. Default value is c(), which would trigger the auto grid search.
dist_matrix	(optional) a matrix that specifies the pair-wise distances between any two objects.
dist_col	(optional) character vector of variable names used for calculating within-pair distance.

<code>exact_col</code>	(optional) character vector, variable names that we want exact matching on; NULL by default.
<code>propensity_col</code>	(optional) character vector, variable names on which to fit a propensity score (to supply a caliper).
<code>pscore_name</code>	(optional) character, giving the variable name for the fitted propensity score.
<code>ignore_col</code>	(optional) character vector of variable names that should be ignored when constructing the internal matching. NULL by default.
<code>max_unmatched</code>	(optional) numeric, the maximum proportion of unmatched units that can be accepted; default is 0.25.
<code>caliper_option</code>	(optional) numeric, the propensity score caliper value in standard deviations of the estimated propensity scores; default is NULL, which is no caliper.
<code>tol</code>	(optional) numeric, tolerance of close match distance; default is 1e-2.
<code>max_iter</code>	(optional) integer, maximum number of iterations to use in searching for penalty combinations that improve the matching; default is 1, where the algorithm searches for one round.
<code>rho_max_factor</code>	(optional) numeric, the scaling factor used in proposal for penalties; default is 10.
<code>max_pareto_search_iter</code>	(optional) numeric, the number of tries to search for the tol that yield pareto optimal solutions; default is 5.

## Details

Matched designs generated by this function are Pareto optimal for the three objective functions. The degree of relative emphasis among the three objectives in any specific solution is controlled by the penalties, denoted by Greek letter rho. Larger values of `exclusion_penalty` corresponds to increased emphasis on retaining treated units (all else being equal), while larger values of `balance_penalty` corresponds to increased emphasis on marginal balance. Additional details:

- Users may either specify their own distance matrix via the `dist_matrix` argument or ask the function to create a robust Mahalanobis distance matrix internally on a set of covariates specified by the `dist_col` argument; if neither argument is specified an error will result. User-specified distance matrices should have row count equal to the number of treated units and column count equal to the number of controls.
- If the `caliper_option` argument is specified, a propensity score caliper will be imposed, forbidding matches between units more than a fixed distance apart on the propensity score. The caliper will be based either on a user-fit propensity score, identified in the input dataframe by argument `pscore_name`, or by an internally-fit propensity score based on logistic regression against the variables named in `propensity_col`. If `caliper_option` is non-NULL and neither of the other arguments is specified an error will result.
- `tol` controls the precision at which the objective functions is evaluated. When matching problems are especially large or complex it may be necessary to increase `tol` in order to prevent integer overflows in the underlying network flow solver; generally this will be suggested in appropriate warning messages.

- While by default tradeoffs are only assessed at penalty combinations provided by the user, the user may ask for the algorithm to search over additional penalty values in order to identify additional Pareto optimal solutions. `rho_max_factor` is a multiplier applied to initial penalties to discover new solutions, and setting it larger leads to wider exploration; similarly, `max_iter` controls how long the exploration routine runs, with larger values leading to more exploration.

## Value

a named list whose elements are: \* `"rhoList"`: list of penalty combinations for each match \* `"matchList"`: list of matches indexed by number

- `"treatmentCol"`: character of treatment variable
- `"covs"`: character vector of names of the variables used for calculating within-pair distance
- `"exactCovs"`: character vector of names of variables that we want exact or close match on \* `"idMapping"`: numeric vector of row indices for each observation in the sorted data frame for internal use
- `"stats"`: data frame of important statistics (total variation distance) for variable on which marginal balance is measured
- `"b.var"`: character, name of variable on which marginal balance is measured \* `"dataTable"`: data frame sorted by treatment value
- `"t"`: a treatment vector
- `"df"`: the original dataframe input by the user
- `"pair_cost1"`: list of pair-wise distance sum using the first distance measure
- `"pair_cost2"`: list of pair-wise distance sum using the second distance measure (left NULL since only one distance measure is used here).
- `"version"`: (for internal use) the version of the matching function called; "Basic" indicates the matching comes from `dist_bal_match` and "Advanced" from `two_dist_match`.
- `"fPair"`: a vector of values for the first objective function; it corresponds to the pair-wise distance sum according to the first distance measure.
- `"fExclude"`: a vector of values for the second objective function; it corresponds to the number of treated units being unmatched.
- `"fMarginal"`: a vector of values for the third objective function; it corresponds to the marginal balanced distance for the specified variable(s).

## See Also

Other main matching function: [two\\_dist\\_match\(\)](#)

## Examples

```
data("lalonde", package="cobalt")
ps_cols <- c("age", "educ", "married", "nodegree", "race")
treat_val <- "treat"
response_val <- "re78"
pair_dist_val <- c("age", "married", "educ", "nodegree", "race")
my_bal_val <- c("race")
```

```

r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
match_result <- dist_bal_match(data=lalonde, treat_col= treat_val,
marg_bal_col = my_bal_val, exclusion_penalty=r1s, balance_penalty=r2s,
dist_col = pair_dist_val,
propensity_col = ps_cols, max_iter=0)

```

---

dummy

*This is a modified version of the function "dummy" from the R package dummies. Original code Copyright (c) 2011 Decision Patterns.*

---

## Description

Change is made to the "model.matrix" function so that the output could be used for the current package.

## Usage

```

dummy(
  x,
  data = NULL,
  sep = "",
  drop = TRUE,
  fun = as.integer,
  verbose = FALSE,
  name = NULL
)

```

## Arguments

x	a data.frame, matrix or single variable or variable name
data	(optional) if provided, x is the name of a column on the data
sep	(optional) the separator used between variable name and the value
drop	(optional) whether to drop unused levels
fun	(optional) function to coerce the value in the final matrix; 'as.integer' by default
verbose	(optional) whether to print the number of variables; FALSE by default
name	(optional) the column name to be selected for converting; NULL by default

---

edgelist2ISM	<i>Change the edgelist to the infinity sparse matrix</i>
--------------	--

---

**Description**

Change the edgelist to the infinity sparse matrix

**Usage**

```
edgelist2ISM(elist)
```

**Arguments**

elist            the vector of the edges

**Value**

the infinity sparse matrix object

---

excludeCosts	<i>Create a skeleton representation of the exclusion edge costs</i>
--------------	---

---

**Description**

Create a skeleton representation of the exclusion edge costs associated with pairings for a given distance and network

**Usage**

```
excludeCosts(net, exclude.penalty = 1)
```

**Arguments**

net            the network structure  
exclude.penalty            (optional) numeric penalty for excluding a treated unit; 1 by default

**Value**

the skeleton with exclusion edge cost

---

extractEdges                      *Extract edges from the network*

---

**Description**

Extract edges from the network

**Usage**

extractEdges(net)

**Arguments**

net                      the network representation

**Value**

the list of edges

---

---

extractSupply                      *Extract the supply nodes from the net*

---

**Description**

Extract the supply nodes from the net

**Usage**

extractSupply(net)

**Arguments**

net                      the network representation

**Value**

the vector of the supply nodes

---

filter_match_result	<i>Filter match result</i>
---------------------	----------------------------

---

**Description**

Filter match result

**Usage**

```
filter_match_result(matching_result, filter_expr)
```

**Arguments**

matching\_result

the matching result object.

filter\_expr

character, the filtering condition based on the summary table returned by get\_rho\_obj.

**Value**

the filtered match result

---

flattenSkeleton	<i>Turns a skeleton representation of edge costs in a network</i>
-----------------	---

---

**Description**

Turns a skeleton representation of edge costs in a network back into the vector representation expected by the optimization routine. See comment on the costSkeleton function for more details.

**Usage**

```
flattenSkeleton(skeleton)
```

**Arguments**

skeleton

the skeleton structure

**Value**

vector representation expected by the optimization routine.

---

generateRhoObj	<i>Penalty and objective values summary</i>
----------------	---

---

**Description**

Helper function to generate a dataframe with matching number, penalty (rho) values, and objective function values.

**Usage**

```
generateRhoObj(matchingResult)
```

**Arguments**

`matchingResult` `matchingResult` object that contains information for all matches.

**Value**

a dataframe that contains objective function values and rho values corresponding coefficients before each objective function.

**See Also**

Other numerical analysis helper functions: [get\\_balance\\_table\(\)](#), [get\\_rho\\_obj\(\)](#), [get\\_unmatched\(\)](#)

---

generate_rhos	<i>Generate rho pairs</i>
---------------	---------------------------

---

**Description**

An internal helper function that generates the set of rho value pairs used for matching. This function is used when exploring the Pareto optimality of the solutions to the multi-objective optimization in matching.

**Usage**

```
generate_rhos(rho1.list, rho2.list)
```

**Arguments**

`rho1.list` a vector of rho 1 values  
`rho2.list` a vector of rho 2 values

**Value**

a vector of (rho1, rho2) pairs

---

getExactOn	<i>Generate a factor for exact matching.</i>
------------	--

---

**Description**

Generate a factor for exact matching.

**Usage**

```
getExactOn(dat, exactList)
```

**Arguments**

dat	dataframe containing all the variables in exactList
exactList	factor of names of the variables on which we want exact or close matching.

**Value**

factor on which to match exactly, with labels given by concatenating labels for input variables.

---

getPropensityScore	<i>Fit propensity scores using logistic regression.</i>
--------------------	---

---

**Description**

Fit propensity scores using logistic regression.

**Usage**

```
getPropensityScore(data, covs)
```

**Arguments**

data	dataframe that contains a column named "treat", the treatment vector, and columns of covariates specified.
covs	factor of column names of covariates used for fitting a propensity score model.

**Value**

vector of estimated propensity scores (on the probability scale).

---

get_balance_table	<i>Generate balance table</i>
-------------------	-------------------------------

---

### Description

The helper function can generate tabular analytics that quantify covariate imbalance after matching. It only works for the 'Basic' version of matching (produced by `dist_bal_match`).

### Usage

```
get_balance_table(
  matching_result,
  cov_list = NULL,
  display_all = TRUE,
  stat_list = c("mean.diffs")
)
```

### Arguments

<code>matching_result</code>	an object returned by the main matching function <code>dist_bal_match</code>
<code>cov_list</code>	(optional) a vector of names of covariates used for evaluating covariate imbalance; NULL by default.
<code>display_all</code>	(optional) a boolean value indicating whether or not to show the data for all possible matches; TRUE by default
<code>stat_list</code>	(optional) a vector of statistics that are calculated for evaluating the covariate imbalance between treated and control group. The types that are supported can be found here: <a href="#">bal.tab</a> .

### Details

The result can be either directly used by indexing into the list, or post-processing by the function `compare_tables` that summarizes the covariate balance information in a tidier table. Users can specify the arguments as follows: \* `cov_list`: if it is set of NULL, all the covariates are included for the covariate balance table; otherwise, only the specified covariates will be included in the tabular result. \* `display_all`: by default, the summary statistics for each match are included when the argument is set to TRUE. If the user only wants to see the summary statistics for matches with varying performance on three different objective values, the function would only display the matches with number of treated units being excluded at different quantiles. User can switch to the brief version by setting the parameter to FALSE. \* `stat_list` is the list of statistics used for measuring balance. The argument is the same as `stats` argument in [bal.tab](#), which is the function that is used for calculating the statistics. By default, only standardized difference in means is calculated.

### Value

a named list object containing covariate balance table and statistics for numer of units being matched for each match; the names are the character of index for each match in the `matchResult`.

**See Also**

Other numerical analysis helper functions: [generateRhoObj\(\)](#), [get\\_rho\\_obj\(\)](#), [get\\_unmatched\(\)](#)

**Examples**

```
## Generate matches
data("lalonde", package="cobalt")
ps_cols <- c("age", "educ", "married", "nodegree", "race")
treat_val <- "treat"
response_val <- "re78"
pair_dist_val <- c("age", "married", "educ", "nodegree", "race")
my_bal_val <- c("race")
r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
match_result <- dist_bal_match(data=lalonde, treat_col= treat_val,
marg_bal_col = my_bal_val, exclusion_penalty=r1s, balance_penalty=r2s,
dist_col = pair_dist_val,
propensity_col = ps_cols, max_iter=0)

## Generate summary table for balance
balance_tables <- get_balance_table(match_result)
balance_tables_10 <- balance_tables$'10'
```

---

get_five_index	<i>An internal helper function that gives the index of matching with a wide range of number of treated units left unmatched</i>
----------------	---

---

**Description**

An internal helper function that gives the index of matching with a wide range of number of treated units left unmatched

**Usage**

```
get_five_index(matching_result)
```

**Arguments**

matching\_result  
an object returned by the main matching function `dist_bal_match`

**Value**

a vector of five matching indices with the number of treated units excluded at 0th, 25th, 50th, 75th and 100th percentiles respectively.

---

`get_pairdist_balance_graph`*Total variation imbalance vs. marginal imbalance*

---

### Description

Plotting function that generate sum of pairwise distance vs. total variation imbalance on specified balance variable. This function only works for 'Basic' version of matching (conducted using `dist_bal_match`).

### Usage

```
get_pairdist_balance_graph(matching_result)
```

### Arguments

`matching_result`  
an object returned by the main matching function `dist_bal_match`

### Value

No return value, called for visualization of match result

### See Also

Other Graphical helper functions for analysis: [get\\_pairdist\\_graph\(\)](#), [get\\_tv\\_graph\(\)](#)

### Examples

```
## Generate matches
data("lalonde", package="cobalt")
ps_cols <- c("age", "educ", "married", "nodegree", "race")
treat_val <- "treat"
response_val <- "re78"
pair_dist_val <- c("age", "married", "educ", "nodegree", "race")
my_bal_val <- c("race")
r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
match_result <- dist_bal_match(data=lalonde, treat_col= treat_val,
marg_bal_col = my_bal_val, exclusion_penalty=r1s, balance_penalty=r2s,
dist_col = pair_dist_val,
propensity_col = ps_cols, max_iter=0)
## Visualize the tradeoff between the pair-wise distance sum and
## total variation distance
get_pairdist_balance_graph(match_result)
```



---

get_rho_obj	<i>Penalty and objective values summary</i>
-------------	---

---

**Description**

Helper function to generate a dataframe with matching number, penalty (rho) values, and objective function values.

**Usage**

```
get_rho_obj(matching_result)
```

**Arguments**

`matching_result`  
matchingResult object that contains information for all matches.

**Value**

a dataframe that contains objective function values and rho values corresponding coefficients before each objective function.

**See Also**

Other numerical analysis helper functions: [generateRhoObj\(\)](#), [get\\_balance\\_table\(\)](#), [get\\_unmatched\(\)](#)

---

get_tv_graph	<i>Marginal imbalance vs. exclusion</i>
--------------	---

---

**Description**

Plotting function that visualizes the tradeoff between the total variation imbalance on a specified variable and the number of unmatched treated units. This function only works for the 'Basic' version of matching (conducted using `dist_bal_match`).

**Usage**

```
get_tv_graph(matching_result)
```

**Arguments**

`matching_result`  
an object returned by the main matching function `dist_bal_match`

**Value**

No return value, called for visualization of match result

**See Also**

Other Graphical helper functions for analysis: [get\\_pairedist\\_balance\\_graph\(\)](#), [get\\_pairedist\\_graph\(\)](#)

**Examples**

```
## Generate matches
data("lalonde", package="cobalt")
ps_cols <- c("age", "educ", "married", "nodegree", "race")
treat_val <- "treat"
response_val <- "re78"
pair_dist_val <- c("age", "married", "educ", "nodegree", "race")
my_bal_val <- c("race")
r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
match_result <- dist_bal_match(data=lalonde, treat_col= treat_val,
marg_bal_col = my_bal_val, exclusion_penalty=r1s, balance_penalty=r2s,
dist_col = pair_dist_val,
propensity_col = ps_cols, max_iter=0)
## Generate visualization of tradeoff between total variation distance and
## number of treated units left unmatched
get_tv_graph(match_result)
```

---

get\_unmatched

*Get unmatched percentage*


---

**Description**

A function that generate the percentage of unmatched units for each match.

**Usage**

```
get_unmatched(matching_result)
```

**Arguments**

```
matching_result
```

matchingResult object that contains information for all matches

**Value**

data frame with three columns, one containing the matching index, one containing the number of matched units, and one containing the percentage of matched units (out of original treated group size).

**See Also**

Other numerical analysis helper functions: [generateRhoObj\(\)](#), [get\\_balance\\_table\(\)](#), [get\\_rho\\_obj\(\)](#)

**Examples**

```
## Not run:  
get_unmatched(match_result)  
  
## End(Not run)
```

---

```
makeInfinitySparseMatrix
```

*Internal helper to build infinity sparse matrix*

---

**Description**

Formats the data and make a call to [InfinitySparseMatrix-class](#)

**Usage**

```
makeInfinitySparseMatrix(  
  data,  
  cols,  
  rows,  
  colnames = NULL,  
  rownames = NULL,  
  dimension = NULL,  
  call = NULL  
)
```

**Arguments**

data	the input numeric vector of cost
cols	the input numeric vector corresponding to control units
rows	the input numeric vector corresponding to treated units
colnames	(optional) vector containing names for all control units
rownames	(optional) vector containing names for all treated units
dimension	(optional) vector of number of treated and control units
call	(optional) funtion call used to create the InfinitySparseMatrix

**Value**

an InfinitySparseMatrix object

---

makeSparse	<i>Helper function to mask edges</i>
------------	--------------------------------------

---

**Description**

Remove some of the treatment-control edges from a network flow representation of a match (forbidding those pairings)

**Usage**

```
makeSparse(net, mask, replaceMask = TRUE)
```

**Arguments**

net	the network object
mask	a matrix indicating whether to exclude the corresponding edge
replaceMask	(optional) whether to mask

**Value**

the masked network structure object

---

matched_data	<i>Get matched dataframe</i>
--------------	------------------------------

---

**Description**

A function that returns the dataframe that contains only matched pairs from the original data frame with specified match index

**Usage**

```
matched_data(matching_result, match_num)
```

**Arguments**

matching_result	an object returned by the main matching function <code>dist_bal_match</code>
match_num	Integer index of match that the user want to extract paired observations from

**Value**

dataframe that contains only matched pair data

**Examples**

```
## Generate Matches
data("lalonde", package="cobalt")
ps_cols <- c("age", "educ", "married", "nodegree", "race")
treat_val <- "treat"
response_val <- "re78"
pair_dist_val <- c("age", "married", "educ", "nodegree", "race")
my_bal_val <- c("race")
r1s <- c(0.01,1,2,4,4.4,5.2,5.4,5.6,5.8,6)
r2s <- c(0.001)
match_result <- dist_bal_match(data=lalonde, treat_col= treat_val,
marg_bal_col = my_bal_val, exclusion_penalty=r1s, balance_penalty=r2s,
dist_col = pair_dist_val,
propensity_col = ps_cols, max_iter=0)
matched_data(match_result, 1)
```

---

matched_index	<i>An internal helper function that translate the matching index in the sorted data frame to the original dataframe's row index</i>
---------------	---

---

**Description**

An internal helper function that translate the matching index in the sorted data frame to the original dataframe's row index

**Usage**

```
matched_index(matchingResult)
```

**Arguments**

matchingResult an object returned by the main matching function dist\_bal\_match

---

matrix2cost	<i>change the distance matrix to cost</i>
-------------	---

---

**Description**

change the distance matrix to cost

**Usage**

```
matrix2cost(net, distance)
```

**Arguments**

net                the network structure  
distance          distance matrix

**Value**

the vector of cost

---

matrix2edgelist      *Helper function to convert matrix to list*

---

**Description**

Convert between a matrix representation of distances between treated and control units and a list of vectors (default format for build.dist.struct function in rebalance package)

**Usage**

```
matrix2edgelist(mat)
```

**Arguments**

mat                matrix representation of distances between treated and control units

**Value**

list of vector representation of distances

---

meldMask            *Helper function to combine two sparse distances*

---

**Description**

Combine two sparse distances, allowing only pairings allowed by both

**Usage**

```
meldMask(mask1, mask2)
```

**Arguments**

mask1              matrix of the first mask  
mask2              matrix of the second mask

**Value**

combined mask structure

---

netFlowMatch	<i>Create network flow structure</i>
--------------	--------------------------------------

---

**Description**

Create network flow structure

**Usage**

```
netFlowMatch(z, IDs = NULL)
```

**Arguments**

z	a vector of treatment vectors
IDs	(optional) the name of the units

**Value**

a networks structure

---

obj.to.match	<i>An internal helper function that transforms the output from the RELAX algorithm to a data structure that is more interpretable for the output of the main matching function</i>
--------------	--

---

**Description**

An internal helper function that transforms the output from the RELAX algorithm to a data structure that is more interpretable for the output of the main matching function

**Usage**

```
obj.to.match(out.elem, already.done = NULL, prev.obj = NULL)
```

**Arguments**

out.elem	a named list whose elements are: (1) the net structure (2) the edge weights of pair-wise distance (3) the edge weights of marginal balance (4) the list of rho value pairs (5) the named list of solutions from the RELAX algorithm
already.done	a factor indicating the index of matches already been transformed
prev.obj	an object of previously transformed matches

**Value**

a named list with elements containing matching information useful for the main matching function

---

pairCosts	<i>Create a skeleton representation of the edge costs</i>
-----------	---

---

**Description**

Create a skeleton representation of the edge costs

**Usage**

```
pairCosts(dist.struct, net)
```

**Arguments**

dist.struct	the distance structure
net	the net structure

**Value**

the skeleton representation of the given distance pairs and the net

---

rho_proposition	<i>Generate penalty coefficient pairs</i>
-----------------	---

---

**Description**

An internal helper function used for automatically generating the set of rho values used for grid search in exploring the Pareto optimal set of solutions.

**Usage**

```
rho_proposition(
  paircosts.list,
  rho.max.factor = 10,
  rho1old,
  rho2old,
  rho.min = 0.01
)
```

**Arguments**

paircosts.list	a vector of pair-wise distance.
rho.max.factor	a numeric value indicating the maximal rho values.
rho1old	a vector of numeric values of rho1 used before.
rho2old	a vector of numeric values of rho2 used before.
rho.min	smallest rho value to consider.

**Value**

a vector of pairs of rho values for future search.

---

solveP	<i>Solve the network flow problem - basic version</i>
--------	---

---

**Description**

Solve the network flow problem - basic version

**Usage**

```
solveP(net, f1.list, f2.list, rho, tol = 1e-05)
```

**Arguments**

net	the network representation
f1.list	the list of the first objective functions values for each node
f2.list	the list of the second objective functions values for each node
rho	the penalty coefficient
tol	the tolerance value for precision

**Value**

the solution represented in a named list

---

solveP1	<i>Solve the network flow problem - twoDistMatch</i>
---------	--

---

**Description**

Solve the network flow problem - twoDistMatch

**Usage**

```
solveP1(net, f1.list, f2.list, f3.list, rho1, rho2 = 0, tol = 1e-05)
```

**Arguments**

net	the network representation
f1.list	the list of the first objective functions values for each node
f2.list	the list of the second objective functions values for each node
f3.list	the list of the third objective functions values for each node
rho1	the penalty coefficient for the second objective
rho2	the penalty coefficient for the third objective
tol	the tolerance value for precision

**Value**

the solution represented in a named list

---

```
summary.multiObjMatch Generate numerical summary
```

---

**Description**

Main summary functions for providing tables of numerical information in matching penalties, objective function values, and balance.

**Usage**

```
## S3 method for class 'multiObjMatch'
summary(
  object,
  type = "penalty",
  cov_list = NULL,
  display_all = TRUE,
  stat = "mean.diff",
  ...
)
```

**Arguments**

object	the matching result returned by either <code>dist_bal_match</code> or <code>two_dist_match</code> .
type	(optional) the type of the summary result in <code>c("penalty", "exclusion", "balance")</code> . When "penalty" is passed in, the objective function values and the penalty values are displayed for each match; when "exclusion" is passed in, the number of units being matched is displayed for each match; when "balance" is passed in, the covariate balance table from <code>bal.tab</code> function in <code>cobalt</code> function is displayed and user can change <code>covList</code> to specify the variables to examine. "penalty" by default.
cov_list	(optional) factor of names of covariates that we want to evaluate covariate balance on if "balance" is passed in for type; default is <code>NULL</code> . When set to <code>NULL</code> , the program will compare the covariates that have been used to construct a propensity model.
display_all	(optional) boolean value of whether to display all the matches if "balance" is passed in for type; default is <code>TRUE</code> , where all matches are displayed.
stat	(optional) character of the name of the statistic used for measuring covariate balance if "balance" is passed in for type; default is "mean.diff". This argument is the same as used in "cobalt" package, see: <a href="#">bal.tab</a>
...	ignored.

**Value**

a summary dataframe of the corresponding type.

---

two\_dist\_match

*Optimal tradeoffs among two distances and exclusion*


---

### Description

Explores tradeoffs among three objective functions in multivariate matching: sums of two different user-specified covariate distances within matched pairs, and the number of treated units included in the match.

### Usage

```
two_dist_match(
  dist1_type = "user",
  dist2_type = "user",
  dist1_matrix = NULL,
  data = NULL,
  dist2_matrix = NULL,
  treat_col = NULL,
  dist1_col = NULL,
  dist2_col = NULL,
  exclusion_penalty = c(),
  dist2_penalty = c(),
  marg_bal_col = NULL,
  exact_col = NULL,
  propensity_col = NULL,
  pscore_name = NULL,
  ignore_col = NULL,
  max_unmatched = 0.25,
  caliper_option = NULL,
  tol = 0.01,
  max_iter = 1,
  rho_max_factor = 10,
  max_pareto_search_iter = 5
)
```

### Arguments

dist1_type	One of ("euclidean", "robust_mahalanobis", "user") indicating the type of distance that are used for the first distance objective functions. NULL by default.
dist2_type	One of ("euclidean", "robust_mahalanobis", "user") character indicating the type of distance that are used for the second distance objective functions. NULL by default.
dist1_matrix	(optional) matrix object that represents the distance matrix using the first distance measure; dist1_type must be passed in as "user" if dist1_matrix is non-empty

data	(optional) data frame that contain columns indicating treatment, outcome and covariates
dist2_matrix	(optional) matrix object that represents the distance matrix using the second distance measure; dist2_type must be passed in as "user" if dist2_matrix is non-empty
treat_col	(optional) character, name of the column indicating treatment assignment.
dist1_col	(optional) character vector names of the variables used for calculating covariate distance using first distance measure specified by dType
dist2_col	(optional) character vector, names of the variables used for calculating covariate distance using second distance measure specified by dType1
exclusion_penalty	(optional) numeric vector, penalty values associated with exclusion. Empty by default, where auto grid search is triggered.
dist2_penalty	(optional) numeric vector, penalty values associated with the distance specified by dist2_matrix or dist2_type. Empty by default, where auto grid search is triggered.
marg_bal_col	(optional) character, column name of the variable on which to evaluate balance.
exact_col	(optional) character vector, names of the variables on which to match exactly; NULL by default.
propensity_col	character vector, names of columns on which to fit a propensity score model.
pscore_name	(optional) character, name of the column containing fitted propensity scores; default is NULL.
ignore_col	(optional) character vector of variable names that should be ignored when constructing the internal matching. NULL by default.
max_unmatched	(optional) numeric, maximum proportion of unmatched units that can be accepted; default is 0.25.
caliper_option	(optional) numeric, the propensity score caliper value in standard deviations of the estimated propensity scores; default is NULL, which is no caliper.
tol	(optional) numeric, tolerance of close match distance; default is 1e-2.
max_iter	(optional) integer, maximum number of iterations to use in searching for penalty combinations that improve the matching; default is 1, where the algorithm searches for one round.
rho_max_factor	(optional) numeric, the scaling factor used in proposal for penalties; default is 10.
max_pareto_search_iter	(optional) numeric, the number of tries to search for the tol that yield pareto optimal solutions; default is 5.

## Details

Matched designs generated by this function are Pareto optimal for the three objective functions. The degree of relative emphasis among the three objectives in any specific solution is controlled by the penalties, denoted by Greek letter rho. Larger values for the penalties associated with the two distances correspond to increased emphasis close matching on these distances, at the possible cost of excluding more treated units. Additional details:

- Users may either specify their own distance matrices (specifying the User option in `dist1_type` and/or `dist2_type` and supplying arguments to `dist1_matrix` and/or `dist2_matrix` respectively) or ask the function to create Mahalanobis or Euclidean distances on sets of covariates specified by the `dist1_col` and `dist2_col` arguments. If `dist1_type` or `dist2_type` is not specified, if one of these is set to user and the corresponding `dist1_matrix` argument is not provided, or if one is NOT set to User and the corresponding `dist1_col` argument is not provided, the code would error out.
- User-specified distance matrices passed to `dist1_matrix` or `dist2_matrix` should have row count equal to the number of treated units and column count equal to the number of controls.
- If the `caliper_option` argument is specified, a propensity score caliper will be imposed, forbidding matches between units more than a fixed distance apart on the propensity score. The caliper will be based either on a user-fit propensity score, identified in the input dataframe by argument `pscore_name`, or by an internally-fit propensity score based on logistic regression against the variables named in `propensity_col`. If `caliper_option` is non-NULL and neither of the other arguments is specified an error will result.
- `tol` controls the precision at which the objective functions is evaluated. When matching problems are especially large or complex it may be necessary to increase `toleranceOption` in order to prevent integer overflows in the underlying network flow solver; generally this will be suggested in appropriate warning messages.
- While by default tradeoffs are only assessed at penalty combinations provided by the user, the user may ask for the algorithm to search over additional penalty values in order to identify additional Pareto optimal solutions. `rho_max_factor` is a multiplier applied to initial penalty values to discover new solutions, and setting it larger leads to wider exploration; similarly, `max_iter` controls how long the exploration routine runs, with larger values leading to more exploration.

## Value

a named list whose elements are:

- "rhoList": list of penalty combinations for each match
- "matchList": list of matches indexed by number
- "treatmentCol": character of treatment variable
- "covs": character vector of names of the variables used for calculating within-pair distance
- "exactCovs": character vector of names of variables that we want exact or close match on
- "idMapping": numeric vector of row indices for each observation in the sorted data frame for internal use
- "stats": data frame of important statistics (total variation distance) for variable on which marginal balance is measured
- "b.var": character, name of variable on which marginal balance is measured (left NULL since no balance constraint is imposed here).
- "dataTable": data frame sorted by treatment value
- "t": a treatment vector
- "df": the original dataframe input by the user
- "pair\_cost1": list of pair-wise distance sum using the first distance measure

- "pair\_cost2": list of pair-wise distance sum using the second distance measure
- "version": (for internal use) the version of the matching function called; "Basic" indicates the matching comes from dist\_bal\_match and "Advanced" from two\_dist\_match.
- "fDist1": a vector of values for the first objective function; it corresponds to the pair-wise distance sum according to the first distance measure.
- "fExclude": a vector of values for the second objective function; it corresponds to the number of treated units being unmatched.
- "fDist2": a vector of values for the third objective function; it corresponds to the pair-wise distance sum corresponds to the

### See Also

Other main matching function: [dist\\_bal\\_match\(\)](#)

### Examples

```
x1 = rnorm(100, 0, 0.5)
x2 = rnorm(100, 0, 0.1)
x3 = rnorm(100, 0, 1)
x4 = rnorm(100, x1, 0.1)
r1ss <- seq(0.1,50, 10)
r2ss <- seq(0.1,50, 10)
x = cbind(x1, x2, x3,x4)
z = sample(c(rep(1, 50), rep(0, 50)))
e1 = rnorm(100, 0, 1.5)
e0 = rnorm(100, 0, 1.5)
y1impute = x1^2 + 0.6*x2^2 + 1 + e1
y0impute = x1^2 + 0.6*x2^2 + e0
treat = (z==1)
y = ifelse(treat, y1impute, y0impute)
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
names(x) <- c("x1", "x2", "x3", "x4")
df <- data.frame(cbind(z, y, x))
df$x5 <- 1
d1 <- as.matrix(dist(df["x1"]))
d2 <- as.matrix(dist(df["x2"]))
idx <- 1:length(z)
treated_units <- idx[z==1]
control_units <- idx[z==0]
d1 <- as.matrix(d1[treated_units, control_units])
d2 <- as.matrix(d2[treated_units, control_units])
match_result_1 <- two_dist_match(data=df, treat_col="z", dist1_matrix=d1,
dist1_type= "User", dist2_matrix=d2,
dist2_type="User", marg_bal_col=c("x5"), exclusion_penalty=r1ss,
dist2_penalty=r2ss,
propensity_col = c("x1"), max_iter = 0,
max_pareto_search_iter = 0)
```

---

 visualize

*Visualize tradeoffs*


---

### Description

Main visualization functions for showing the tradeoffs between two of the three objective functions. A 3-d plot can be visualized where the third dimension is represented by coloring of the dots.

### Usage

```
visualize(
  matching_result,
  x_axis = "dist1",
  y_axis = "dist2",
  z_axis = NULL,
  xlab = NULL,
  ylab = NULL,
  zlab = NULL,
  main = NULL,
  display_all = FALSE,
  cond = NULL,
  xlim = NULL,
  ylim = NULL,
  display_index = TRUE,
  average_cost = FALSE
)
```

### Arguments

<code>matching_result</code>	the matching result returned by either <code>dist_bal_match</code> or <code>two_dist_match</code> .
<code>x_axis</code>	character, naming the objective function shown on x-axis; one of ("pair", "marginal", "dist1", "dist2", "exclude", "distance_penalty", "balance_penalty", "dist1_penalty", "dist2_penalty", "exclusion_penalty"), "dist1" by default.
<code>y_axis</code>	character, naming the objective function shown on y-axis; one of ("pair", "marginal", "dist1", "dist2", "exclude", "distance_penalty", "balance_penalty", "dist1_penalty", "dist2_penalty", "exclusion_penalty"), "dist1" by default.
<code>z_axis</code>	character, naming the objective function for coloring; one of ("pair", "marginal", "dist1", "dist2", "exclude"), "exclude" by default.
<code>xlab</code>	(optional) the axis label for x-axis; NULL by default.
<code>ylab</code>	(optional) the axis label for y-axis; NULL by default.
<code>zlab</code>	(optional) the axis label for z-axis; NULL by default.
<code>main</code>	(optional) the title of the graph; NULL by default.

<code>display_all</code>	(optional) whether to show all the labels for match index; FALSE by default, which indicates the visualization function only labels matches at quantiles of number of treated units being excluded.
<code>cond</code>	(optional) NULL by default, which denotes all the matches are shown; otherwise, takes a list of boolean values indicating whether to include each match
<code>xlim</code>	(optional) NULL by default; function automatically takes the max of the first objective function values being plotted on x-axis; if specified otherwise, pass in the numeric vector <code>c(lower_bound, upper_bound)</code>
<code>ylim</code>	(optional) NULL by default; function automatically takes the max of the first objective function values being plotted on y-axis; if specified otherwise, pass in the numeric vector <code>c(lower_bound, upper_bound)</code>
<code>display_index</code>	(optional) TRUE by default; whether to display match index
<code>average_cost</code>	(optional) FALSE by default; whether to show mean cost

### Details

By default, the plotting function will show the tradeoff between the first distance objective function and the marginal balance (if `dist_bal_match`) is used; or simply the second distance objective function, if `two_dist_match` is used.

### Value

No return value, called for visualization of match result

# Index

- \* **Graphical helper functions for analysis**
  - get\_pairdist\_balance\_graph, 24
  - get\_pairdist\_graph, 25
  - get\_tv\_graph, 26
- \* **main matching function**
  - dist\_bal\_match, 13
  - two\_dist\_match, 36
- \* **numerical analysis helper functions**
  - generateRhoObj, 20
  - get\_balance\_table, 22
  - get\_rho\_obj, 26
  - get\_unmatched, 27
- addBalance, 3
- addExclusion, 3
- bal.tab, 8, 22, 35
- balanceCosts, 4
- build.dist.struct, 4
- build.dist.struct\_user, 5
- callrelax, 6
- check\_representative, 6
- combine\_dist, 7
- combine\_match\_result, 7
- compare\_matching, 8
- compare\_tables, 9
- convert\_index, 10
- convert\_names, 10
- costSkeleton, 11
- data\_precheck, 11
- descr.stats\_general, 12
- dist\_bal\_match, 13, 39
- distanceFunctionHelper, 12
- dummy, 16
- edgelist2ISM, 17
- excludeCosts, 17
- extractEdges, 18
- extractSupply, 18
- filter\_match\_result, 19
- flattenSkeleton, 19
- generate\_rhos, 20
- generateRhoObj, 20, 23, 26, 27
- get\_balance\_table, 20, 22, 26, 27
- get\_five\_index, 23
- get\_pairdist\_balance\_graph, 24, 25, 27
- get\_pairdist\_graph, 24, 25, 27
- get\_rho\_obj, 20, 23, 26, 27
- get\_tv\_graph, 24, 25, 26
- get\_unmatched, 20, 23, 26, 27
- getExactOn, 21
- getPropensityScore, 21
- InfinitySparseMatrix-class, 28
- makeInfinitySparseMatrix, 28
- makeSparse, 29
- matched\_data, 29
- matched\_index, 30
- matrix2cost, 30
- matrix2edgelist, 31
- meldMask, 31
- netFlowMatch, 32
- obj.to.match, 32
- pairCosts, 33
- rho\_proposition, 33
- solveP, 34
- solveP1, 34
- summary.multiObjMatch, 35
- two\_dist\_match, 15, 36
- visualize, 40