Package 'metaheuristics'

December 7, 2023

Type Package

Title Helps Melissa avoid tedious metaheuristics work

Version 0.1.0

Author Melissa Klocinski

Maintainer Not relevant but me <mklocinski@gmail.com>

Description Package used to automate metaheuristic work.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Imports magrittr, mhHelper

R topics documented:

`adaptive_memory` *Apply adaptive memory*

Description

Apply an adaptive memory to a list of prior solutions.

Usage

```
adaptive_memory(search_history, memory)
```

Arguments

`search_history`	a list of prior solutions.
`memory`	the maximum length of search_history

Details

Certain heuristics, like tabu search, have an adaptive memory that maintains a search history of a specified length. This function prunes a list of prior solutions to a specified length.

Value

a list of prior solutions.

Examples

```
adaptive_memory(list("010", "210", "100", "120"), 3)
```

`atc_job_flow` *ATC Job Flow*

Description

Pending

Usage

```
atc_job_flow(first_job, jobs, machines, process_times, due_dates, weights, K)
```

Details

Pending

`atc_rank` *ATC Index Function*

Description

Pending

Usage

```
atc_rank(t, p, mp, d, w, K)
```

Details

Pending

binary_max *Calculate maximum feasible value of binary*

Description

Get maximum possible value for based on the length of a binary.

Usage

```
binary_max(binary_sol)
```

Arguments

binary_sol	a list of Boolean values corresponding to a binary string.

Details

A 2-bit binary vector will have a maximum value of 3. A 3-bit binary vector will have a maximum value of 7.

Value

a number.

Examples

```
binary_max(list("1", "1", "0", "0", "1"))
```

binary_round *Binary Round*

Description

Generates round of binary strings.

Usage

```
binary_round(init_sol, neighbors, tabu = list())
```

Arguments

init_sol	A string representing the initial solution.
neighbors	A number specifying the number of neighbors to iterate through.
tabu	(Optional) A tabu list.

Details

Pending.

Value

a list containing a list of the updated binary lists, a list of the updated binary lists converted to strings, a list of the updated binary strings converted to numbers, and a list of the positions that were flipped.

---

binary_to_number *Convert binary string to number*

Description

Translates a binary list to a number.

Usage

```
binary_to_number(binary_sol)
```

Arguments

binary_sol	a list of Boolean values corresponding to a binary string.

Details

Binary lists are lists of Boolean values that represent a binary string. For example, the binary list list(F, T, F, T, T) represents the binary vector 01011.

Binary strings are translated to binary lists using 'string_to_binary()'.

Value

a number.

---

binary_to_string *Convert binary list to string*

Description

Translate binary list to string.

Usage

```
binary_to_string(binary_sol)
```

Arguments

binary_sol	a list of Boolean values corresponding to a binary vector.

Details

Binary lists are lists of Boolean values that represent a binary string. For example, the binary list list(F, T, F, T, T) represents the binary vector 01011.

This function is used to format algorithm output.

Value

a string.

`conduct_flow_jobs` *Job Shop - Flow*

Description

Conducts the entire flow job shop process.

Usage

```
conduct_flow_jobs(param_list)
```

Arguments

`param_list`	a list of job shop parameters, including (and in this order) the initial solution string; job IDs p (processing time), d (due date), and w (weight for delays); number of machines.

Details

Pending

Does not provide an optimal solution.

Value

a list object that contains job completion information for each job and the weighted delays

---

`countoff` *Count off items in a list into queues*

Description

Sorts items into queues based on item position in item list.

Usage

```
countoff(items, queues)
```

Arguments

`items`	a list of items to sort.
`queues`	a list of queues to sort items into.

Details

Works the same dealing n cards to m people, except that it allows for remainders to be included (so the mth player can get (n/m) + 1 cards).

To sort items into queues based on queue length, use 'next_in_line' function.

Value

a list containing item order per queue.

`data_1_mach_n_jobs` *Set Covering Data*

Description

Data to use in one machine, n jobs job shop problems.

Usage

```
data_1_mach_n_jobs
```

Format

a matrix.

Source

OR 670 - Lecture 2 Excel examples.

`data_2_mach_n_jobs` *Set Covering Data*

Description

Data to use in two machines, n jobs job shop problems.

Usage

```
data_2_mach_n_jobs
```

Format

a matrix.

Source

OR 670 - Lecture 2 Excel examples.

data_set_covering *Set Covering Data*

Description

Data to use in set covering problems.

Usage

```
data_set_covering
```

Format

a matrix.

Source

OR 670 - Lecture 2 Excel examples.

data_tsp *Traveling Salesman*

Description

Data in traveling salesman problems.

Usage

```
data_tsp
```

Format

a matrix.

Source

OR 670 - Lecture 2 Excel examples.

elapsed_time *Check elapsed time*

Description

Checks elapsed time against stopping criteria.

Usage

```
elapsed_time(sc_list, stop_crit)
```

Arguments

sc_list	A list of algorithm output that can be evaluated by stopping functions.
stop_crit	The value to be used when determining if an algorithm should stop.

Details

Pending.

Value

a Boolean value.

exchange_values *Exchange values in a list*

Description

Swap values in solution list.

Usage

```
exchange_values(sol_list, one, two)
```

Arguments

sol_list	a solution list.
one	the first position in the solution list that will be exchanged.
two	the second position in the solution list that will be exchanged.

Details

Pending.

Value

a solution list.

flip_positions *Flip values in binary list*

Description

Generate list of positions to flip.

Usage

```
flip_positions(sol_length, neighbors)
```

Arguments

sol_length	the length of a solution list.
neighbors	the number of positions to flip in the solution list.

Details

Pending.

Value

a list of position pairs to flip.

Examples

```
flip_positions(5, 2)
```

genetic_algorithm *Genetic Algorithm*

Description

Executes the genetic algorithm.

Usage

```
genetic_algorithm(
  genes,
  crossover_probability,
  crossover_points,
  mutation_rate,
  elite_parents = 0,
  initial_gen,
  fitness_function,
  fitness_function_params = NULL,
  selection_function,
  crossover_function,
  mutation_function,
  stop_func,
```

```
  stop_crit,
  permutation = F
)
```

Details

Pending.

`job` *job*

Description

Creates a single job for a job shop process.

Usage

```
job(job_info)
```

Arguments

`job_info`	a list of job information.

Details

This function takes in a list of job information that includes p (job length), d (due date), and w (weight for delays).

Value

a list (job object) to be used by 'job_shop'.

`job_assignment` *Job Assignment Problem*

Description

Calculates total set up times for given job assignments.

Usage

```
job_assignment(param_list)
```

Details

Pending

job_shop *Job Shop Problem*

Description

Conducts the entire job shop process, from job object creation to total weighted delay calculations.

Usage

```
job_shop(param_list)
```

Arguments

param_list	a list of job shop parameters, including (and in this order) the initial solution string; a matrix of job information with column order job ID, p (processing time), d (due date), and w (weight for delays); number of machines; and job shop type in a string.

Details

Given a parameter list of job shop information, creates job objects and sends them through the user-specified job shop type function.

Does not provide an optimal solution.

Value

a list object that contains the total process time (machine_time) and the total weighted delay (total_weighted_delay).

k_swap_round *K-Swap Round*

Description

Generates round of permuted strings.

Usage

```
k_swap_round(init_sol, neighbors, tabu = list())
```

Arguments

init_sol	A string representing the initial solution.
neighbors	A number specifying the number of neighbors to iterate through.
tabu	(Optional) A tabu list.

Details

Pending.

Value

a list containing a list of the updated string lists, a list of the updated lists converted to strings, a list of the updated strings converted to function input, and a list of the positions that were flipped.

`local_search` *Local Search*

Description

Executes the local search algorithm and returns the results of the best round.

Usage

```
local_search(
  obj_func,
  problem_params,
  sol_represent,
  eval_func,
  stop_func,
  initial_solution,
  neighbors,
  stop_crit
)
```

Arguments

`obj_func`	The objective function.
`problem_params`	A list of all parameters required to run the objective function.
`sol_represent`	A function that outputs the required solution representation.
`eval_func`	A function that evaluates the algorithm's output.
`stop_func`	A function that evaluates the heuristic's metadata to determine if the search should end.
`initial_solution`	A string representing the initial solution.
`neighbors`	A number representing the number of neighbors to iterate through.
`stop_crit`	A number used by the stop_func function to determine if the search should stop.

Details

Pending.

Value

a list of heuristic output including time to execute the round, round ID, the current round's solution, the current round's function output, the previous round's function output, and the improvement over the last round.

`local_search_storage` *Local Search (Storage)*

Description

Executes the local search algorithm and returns the results of all rounds.

Usage

```
local_search_storage(
  obj_func,
  problem_params,
  sol_represent,
  eval_func,
  stop_func,
  initial_solution,
  neighbors,
  stop_crit
)
```

Arguments

`obj_func`	The objective function.
`problem_params`	A list of all parameters required to run the objective function.
`sol_represent`	A function that outputs the required solution representation.
`eval_func`	A function that evaluates the algorithm's output.
`stop_func`	A function that evaluates the heuristic's metadata to determine if the search should end.
`initial_solution`	A string representing the initial solution.
`neighbors`	A number representing the number of neighbors to iterate through.
`stop_crit`	A number used by the stop_func function to determine if the search should stop.

Details

Pending.

Value

a table of heuristic output including time to execute each round, round ID, each round's solution, each round's function output, each previous round's function output, and the improvement between each round.

`matrix_setup` *Matrix setup*

Description

Creates a named, valid transition matrix from input.

Usage

```
matrix_setup(data, state_names = list(), row_sum_check = T)
```

Arguments

`data`	a list of rbind-ed rows.
`state_names`	(Optional) a list of row and column names list(list(row_names), list(column_names)). If not provided, uses alphabetical labels.
`row_sum_check`	(Optional) Boolean; should the rows sum to one? Default is True.

Details

Used to set up matrices.

Value

a list of prior solutions.

`max_val` *Choose maximum objective function output value*

Description

Identify the solution that yields the max value from a list of solutions.

Usage

```
max_val(xs, ys)
```

Arguments

`xs`	A list of objective function solutions (x values) in its raw form (a list of values).
`ys`	A list of objective function outputs (y values).

Details

Pending.

Value

a list containing the nest solution or the yielding the maximum objective function output in its raw form (a list of values), a cleaned version of the best solution, the solution's value, and the position of the best solution in the solution list.

min_val *Choose minimum objective function output value*

Description

Identify the solution that yields the min value from a list of solutions.

Usage

```
min_val(xs, ys)
```

Arguments

xs	A list of objective function solutions (x values) in its raw form (a list of values).
ys	A list of objective function outputs (y values).

Details

Pending.

Value

a list containing the nest solution or the yielding the minimum objective function output in its raw form (a list of values), a cleaned version of the best solution, the solution's value, and the position of the best solution in the solution list.

m_machines_n_jobs *m machines, n jobs*

Description

Conducts an m machine, n jobs process.

Usage

```
m_machines_n_jobs(jobs, machines, min_max = 1)
```

Arguments

jobs	a list of job objects.
machines	the number of machines in the process.
min_max	(Optional) Used to specify whether the next-queue decision should be based on maximum (default) or minimum total queue value.

Details

Requires a list of job objects (i.e. list of lists of job information) generated by 'job()'. Uses function 'next_in_line()' to determine which queue the next job should be sent to.

Value

a list object that contains the total process time (machine_time) and the total weighted delay (total_weighted_delay).

`m_machines_n_jobs_flow` *m machines, n jobs (flow)*

Description

Conducts an m machine, n jobs process in which the jobs enter the i+1th machine after completing in the ith machine.

Usage

```
m_machines_n_jobs_flow(jobs, machines, min_max = 1)
```

Arguments

`jobs`	a list of job objects.
`machines`	the number of machines in the process.
`min_max`	(Optional) Used to specify whether the next-queue decision should be based on maximum (default) or minimum total queue value.

Details

Requires a list of job objects (i.e. list of lists of job information) generated by 'job()'.

Value

a list object that contains the total process time (machine_time) and the total weighted delay (total_weighted_delay).

`next_in_line` *Sort items into queues based on queue status*

Description

Next in line (used for sorting items to queues with evaluation).

Usage

```
next_in_line(items, queues, min_max = 1)
```

Arguments

`items`	a list of items to sort.
`queues`	a list of queues to sort items into.
`min_max`	(Optional) Default value 1 instructs function to evaluate based on maximum value, -1 instructs function to evaluate based on minimum value.

Details

Pending.

Value

a list containing item order per queue.

Examples

```
next_in_line(items = list(4, 50, 6, 3, 1, 9), queues = list(1, 2))
```

`node_connection_exists`

Check if node connection exists

Description

Checks for connection between two nodes.

Usage

```
node_connection_exists(pair1, pair2)
```

Arguments

`pair1`	the c(row, column) of a matrix cell.
`pair2`	the c(row, column) of a matrix cell.

Details

Compares row, column information of two nodes to determine connection.

Value

a Boolean value.

Examples

```
node_connection_exists(c(1, 3), c(3, 1))
```

node_paths *Generate feasible paths through nodes*

Description

Returns matrix of connected edges.

Usage

```
node_paths(pairs)
```

Arguments

pairs	a list of pairs.

Details

Pending.

Value

a matrix.

number_of_rounds *Check number of rounds*

Description

Checks number of rounds against stopping criteria.

Usage

```
number_of_rounds(sc_list, stop_crit)
```

Arguments

sc_list	A list of algorithm output that can be evaluated by stopping functions.
stop_crit	The value to be used when determining if an algorithm should stop.

Details

Pending.

Value

a Boolean value.

one_machine_n_jobs *one machine, n jobs*

Description

Conducts a one machine, n jobs process.

Usage

```
one_machine_n_jobs(jobs, machines = 1, delay = 0)
```

Arguments

jobs	a list of job objects.
machines	(Optional) number of machines; default is obviously 1
delay	(Optional) used if a delay needs to be added to the whole process.

Details

Requires a list of job objects (i.e. list of lists of job information) generated by 'job()'.

Value

a list object that contains the total process time (machine_time) and the total weighted delay (total_weighted_delay).

permutation_round *Permutation Round*

Description

Generates round of permuted strings.

Usage

```
permutation_round(init_sol, neighbors, tabu = list())
```

Arguments

init_sol	A string representing the initial solution.
neighbors	A number specifying the number of neighbors to iterate through.
tabu	(Optional) A tabu list.

Details

Pending.

Value

a list containing a list of the updated string lists, a list of the updated lists converted to strings, a list of the updated strings converted to function input, and a list of the positions that were flipped.

permute_positions *Permute values in list*

Description

Generate list of positions to permute.

Usage

```
permute_positions(sol_length, neighbors)
```

Arguments

sol_length	the length of a solution list.
neighbors	the number of positions to flip in the solution list.

Details

Pending.

Value

a list of position pairs to flip.

Examples

```
permute_positions(5, 2)
```

relative_improvement *Check relative improvement*

Description

Checks improvement between rounds against stopping criteria.

Usage

```
relative_improvement(sc_list, stop_crit)
```

Arguments

sc_list	A list of algorithm output that can be evaluated by stopping functions.
stop_crit	The value to be used when determining if an algorithm should stop.

Details

Pending.

Value

a Boolean value.

set_covering *Set Covering Problem*

Description

Calculates value of a specific solution to a set covering problem.

Usage

```
set_covering(param_list)
```

Arguments

param_list	a list of job shop parameters, including (and in this order) the initial solution string; a matrix of set covering problem information; the maximum node value for inclusion in the set; the penalty to apply per uncovered node.

Details

Creates a list of nodes that meet the minimum value criteria (valid), creates a path through the valid nodes, then calculates the value of the valid nodes.

Does not provide an optimal solution.

Value

the value of the valid nodes.

string_list_to_bool *Convert list of strings to list of Boolean*

Description

Translate list of 0s and 1s, stored as characters, to a list of Boolean values.

Usage

```
string_list_to_bool(str_list)
```

Arguments

str_list	a list of characters representing a binary string.

Details

Binary lists are initially entered as strings and need to be translated to a list of Boolean values for use by this package. The function 'string_to_list()' can convert a string to a list of characters, which can then be converted to a list of Boolean values by this function.

Value

a list of Boolean values corresponding to a binary string.

Examples

```
string_list_to_bool(list("1", "1", "0", "0", "1"))
```

`string_to_bool` *Convert string to Boolean*

Description

Translates a solution string to a binary list.

Usage

```
string_to_bool(str_sol)
```

Arguments

`str_sol`	a string of characters.

Details

Solution strings can represent sequence information or a binary string. Binary lists are lists of Boolean values that represent a binary string. For example, the binary list list(F, T, F, T, T) represents the binary vector 01011.

Value

a list of Boolean values corresponding to a binary string.

Examples

```
string_to_bool("2143")
```

`string_to_list` *Convert string to list*

Description

Translates a solution string to list.

Usage

```
string_to_list(str_sol)
```

Arguments

`str_sol`	a string of characters.

Details

Solution strings represent sequence information.

Value

a list of characters.

Examples

```
string_to_list("2143")
```

`tabu_search` *Tabu Search*

Description

Executes the tabu search algorithm and returns the results of all rounds.

Usage

```
tabu_search(
  obj_func,
  problem_params,
  sol_represent,
  eval_func,
  stop_func,
  initial_solution,
  neighbors,
  stop_crit,
  memory_length
)
```

Arguments

`obj_func`	The objective function.
`problem_params`	A list of all parameters required to run the objective function.
`sol_represent`	A function that outputs the required solution representation.
`eval_func`	A function that evaluates the algorithm's output.
`stop_func`	A function that evaluates the heuristic's metadata to determine if the search should end.
`initial_solution`	A string representing the initial solution.
`neighbors`	A number representing the number of neighbors to iterate through.
`stop_crit`	A number used by the stop_func function to determine if the search should stop.
`memory_length`	A number specifying the length of the tabu list.

Details

Pending.

Value

a table of heuristic output including time to execute each round, round ID, the round's tabu list, each round's solution, each round's function output, each previous round's function output, and the improvement between each round.

`tabu_search_storage` *Tabu Search (Storage)*

Description

Executes the tabu search algorithm and returns the results of all rounds.

Usage

```
tabu_search_storage(
  obj_func,
  problem_params,
  sol_represent,
  eval_func,
  stop_func,
  initial_solution,
  neighbors,
  stop_crit,
  memory_length
)
```

Arguments

`obj_func`	The objective function.
`problem_params`	A list of all parameters required to run the objective function.
`sol_represent`	A function that outputs the required solution representation.
`eval_func`	A function that evaluates the algorithm's output.
`stop_func`	A function that evaluates the heuristic's metadata to determine if the search should end.
`initial_solution`	A string representing the initial solution.
`neighbors`	A number representing the number of neighbors to iterate through.
`stop_crit`	A number used by the stop_func function to determine if the search should stop.
`memory_length`	A number specifying the length of the tabu list.

Details

Pending.

Value

a table of heuristic output including time to execute each round, round ID, the round's tabu list each round's solution, each round's function output, each previous round's function output, and the improvement between each round.

traveling_salesman *Traveling Sales Problem*

Description

Calculates total distance traveled in traveling salesman problem given specific node order and matrix of edge values.

Usage

```
traveling_salesman(solution_list, round_trip = T)
```

Arguments

solution_list	a string of characters.
round_trip	(Optional) Boolean; is the path a round trip or does it end without returning.

Details

This function does not find the optimal path.

Value

the total distance traveled.

update_binary *Update Binary List*

Description

Flips a Boolean value in a binary list.

Usage

```
update_binary(binary_sol, position)
```

Arguments

binary_sol	A binary list, or list of Boolean values.
position	A number or vector of numbers specifying the position(s) in the binary list to be flipped.

Details

Pending.

Value

a binary list.

Index