# Package 'heumilkr'

November 13, 2026

**Title** Heuristic Capacitated Vehicle Routing Problem Solver

**Version** 0.4.0

**Description**
Implements the Clarke-Wright algorithm to find a quasi-optimal solution to the Capacitated Vehicle Routing Problem. See Clarke, G. and Wright, J.R. (1964) <doi:10.1287/opre.12.4.568> for details. Sites can also have goods picked up and brought back to the depot (``backhaul''). The implementation is accompanied by helper functions to inspect its solution.

**License** GPL (>= 3)

**Encoding** UTF-8

**LinkingTo** cpp11

**SystemRequirements** C++17

**URL** https://github.com/lschneiderbauer/heumilkr,
https://lschneiderbauer.github.io/heumilkr/

**BugReports** https://github.com/lschneiderbauer/heumilkr/issues

**Imports** rlang (>= 1.1.0), cli (>= 3.6.0), xml2 (>= 1.3.0), ggplot2 (>= 3.4.0)

**Suggests** testthat (>= 3.3.0), hedgehog (>= 0.1), ggExtra (>= 0.10.0), scales (>= 1.3.0), curl, knitr, rmarkdown

**Config/testthat/edition** 3

**Depends** R (>= 3.5.0)

**LazyData** true

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.1.1

**NeedsCompilation** yes

**Author** Lukas Schneiderbauer [aut, cre, cph]

**Maintainer** Lukas Schneiderbauer <lukas.schneiderbauer@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-11-13 09:10:33 UTC

# Contents

autoplot.heumilkr_solution . . . 2
clarke_wright . . . 3
clarke_wright_cvrplib . . . 5
cvrplib_A . . . 6
cvrplib_B . . . 6
cvrplib_download . . . 7
cvrplib_E . . . 8
cvrplib_F . . . 8
cvrplib_ls . . . 9
cvrplib_Tai . . . 9
milkr_cost . . . 10
milkr_saving . . . 11
plot.heumilkr_solution . . . 12

**Index** **13**

---

`autoplot.heumilkr_solution`

*Create ggplot for a CVRP solution*

---

## Description

Represents the sites and runs on a 2D plane so that the distances between sites on the drawn 2D plane correspond to `distances` provided to the solver `clarke_wright()`.

The individual runs are distinguished by color. The site locations are marked with round circles while the depot is depicted as a square. Delivery and pickup sites are not distinguished. The line types (solid/dashed/...) are associated to different vehicle types.

## Usage

```
## S3 method for class 'heumilkr_solution'
autoplot(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A "`heumilkr_solution`" object, typically obtained by `clarke_wright()`. |
| `...` | Not used. |

## Details

Distance information between sites only determine site positions on a 2D plane up to rotations and translations: those are fixed arbitrarily.

## Value

A ggplot object.

clarke_wright *Clarke-Wright algorithm, a Capacitated Vehicle Routing Problem solver*

## Description

Finds a quasi-optimal solution to the Capacitated Vehicle Routing Problem (CVRP). All runs start and end at a single depot (origin). Goods are delivered from the depot to sites with positive demand, and picked up from sites with negative demand and brought back to the depot ("backhaul").

## Usage

```
clarke_wright(
  demand,
  distances,
  vehicles,
  restrictions = data.frame(vehicle = integer(), site = integer())
)
```

## Arguments

| | |
|---|---|
| demand | A `numeric()` vector consisting of "demands" indexed by sites. The `i`th entry refers to the demand of the `i`th site. The length of the vector equals the number of sites `N`. The units of demand values need to match the units of vehicle capacity values. Missing or infinite values are not allowed. Positive values are goods to be delivered from the depot to a site. Negative values are goods to be picked up at a site and brought to the depot, i.e. "backhaul". Sites with positive and negative demands can be visited in any order within a run, as long as the load stays within the vehicle capacity at every point of the run. Goods picked up at one site can not be delivered to another site. Sites with a demand of zero are still visited. |
| distances | An object of class `dist`, created by `stats::dist()`, with `(N + 1)` locations describing the distances between individual sites. The first index refers to the depot. The `(i+1)`th index refers to the `i`th site (as defined by `demand`). |
| vehicles | A `data.frame()` describing available vehicle types and their respective capacities. One row per vehicle type. The data frame is expected to have two columns:<br>• `n` - Number of available vehicles, a whole number that is not negative. This can be set to `NA` if the number is "infinite" (i.e. effectively the maximal integer value on your machine.). It is recommended to keep at least one vehicle type as "infinite", otherwise the solver might raise a run time error due to initially not having enough vehicles available (even though the final solution might satisfy the availability restrictions).<br>• `caps` - The vehicle capacity in same units as `demand`, positive and finite.<br>The order of the `data.frame()` is relevant and determines the prioritization of vehicle assignments to runs (in case two or more vehicle types are eligible for assignment the "first" vehicle is chosen). In a typical scenario "more expensive" |

vehicles should be further down in the list (so the cheaper one is chosen in case there is doubt). Since higher capacity vehicles usually involve higher costs sorting the data frame by capacity is usually a good rule of thumb.

`restrictions` An optional `data.frame()` that allows to define vehicle type restrictions for particular sites in the form of a blacklist. The data frame is expected to have two columns:

- `vehicle` - The vehicle type index as a whole number, starting at 0 (i.e. 0 refers to the first row of `vehicles`).
- `site` - The site index as a whole number, starting at 0 (i.e. 0 refers to the first entry of `demand`).

Each row defines a restriction: vehicle type `vehicle` can not approach site `site`. Defaults to an empty `data.frame()`, i.e. no restrictions are enforced.

## Details

See the original paper, Clarke, G. and Wright, J.R. (1964) doi:10.1287/opre.12.4.568, for a detailed explanation of the Clarke-Wright algorithm.

To support deliveries and pickups in any order within a run, two runs can only be merged by connecting the end of one run to the start of the other. The original algorithm can connect either end of a run, but reversing a run with both deliveries and pickups would change its load along the way.

As an additional step after merging, the visiting order is re-optimized with a greedy heuristic for runs that either only deliver or only pick up. Runs with both keep their order, so that the load stays within the vehicle capacity.

## Value

Returns a `"heumilkr_solution"` object, a named `list()` of three `data.frame()`s with names "runs", "sites" and "visits", bestowed with additional attributes.

- `runs`: Each record reflects a single run/tour.
  - `run` - A run identifier.
  - `vehicle` - The vehicle type index associated to the run, starting at 0 (i.e. 0 refers to the first row of `vehicles`).
  - `max_load` - The maximal load of the vehicle on this run. It is always smaller or equal the vehicle capacity.
  - `distance` - The travel distance of the particular run.
- `sites`: Each record reflects a site. Its columns consist of:
  - `site` - The site index, starting at 0 (i.e. 0 refers to the first entry of `demand`).
  - `demand` - The site demand. This is supplied as an input.
- `visits`: Each record reflects a run visit of a particular site. It is implied that each run starts and ends at the origin. Its columns consist of:
  - `run` - Identifies the run the site is assigned to.
  - `site` - The site index associated to the run, starting at 0 (i.e. 0 refers to the first entry of `demand`).
  - `order` - Integer values providing the visiting order within each run.

- `load` - The load after visiting site `site` in units of demand per particular run.

Unless a site demand exceeds the vehicle capacities it is always assigned to only a single run. Otherwise it is split across multiple runs, and the site appears once per run in `visits`. `load` and `max_load` then refer to the share of the demand served by the particular run, while `demand` in `sites` still holds the full demand.

## Examples

```
demand <- c(3, 2, 4, 2)

positions <-
  data.frame(
    pos_x = c(0, 1, -1, 2, 3),
    pos_y = c(0, 1, 1, 2, 3)
  )

clarke_wright(
  demand,
  dist(positions),
  data.frame(n = NA_integer_, caps = 6)
)
```

---

# clarke_wright_cvrplib *Apply* clarke_wright() *to CVRPLIB data*

---

## Description

Apply `clarke_wright()` to CVRPLIB data

## Usage

```
clarke_wright_cvrplib(instance)
```

## Arguments

| | |
|---|---|
| `instance` | A "cvrplib_instance" object. See `cvrplib_download()` or bundled CVRPLIB data like cvrplib_A. |

## Value

A "heumilkr_solution" object. See `clarke_wright()`.

## See Also

Other cvrplib: `cvrplib_download()`, `cvrplib_ls()`

## Examples

```
clarke_wright_cvrplib(cvrplib_A[[1]])
```

---

`cvrplib_A` *CVRP instance data by Augerat, 1995*

---

## Description

A collection of CVRP instances by Augerat, 1995, provided courtesy of CVRPLIB. See CVRPLIB for visualizations of the instances and their solutions as well as a multitude of alternative instance data.

## Usage

```
cvrplib_A
```

## Format

`cvrplib_A`:
A list of CVRP instances as "`cvrplib_instance`" objects. The instances can be directly fed into solver algorithm, e.g. via `clarke_wright_cvrplib()`.

## Source

https://galgos.inf.puc-rio.br/cvrplib/en/instances

---

`cvrplib_B` *CVRP instance data by Augerat, 1995*

---

## Description

A collection of CVRP instances by Augerat, 1995, provided courtesy of CVRPLIB. See CVRPLIB for visualizations of the instances and their solutions as well as a multitude of alternative instance data.

## Usage

```
cvrplib_B
```

## Format

`cvrplib_B`:
A list of CVRP instances as "`cvrplib_instance`" objects. The instances can be directly fed into solver algorithm, e.g. via `clarke_wright_cvrplib()`.

## Source

https://galgos.inf.puc-rio.br/cvrplib/en/instances

---

`cvrplib_download` *CVRPLIB problem instance downloader*

---

## Description

CVRPLIB offers a selection of CVRP problem instances. This function downloads the instance data and conveniently makes it available to be fed into solver functions, e.g. with `clarke_wright_cvrplib()`. The primary purpose for those instances is benchmarking / comparing speed as well as performance of solvers.

## Usage

```
cvrplib_download(qualifier, cvrplib_url = "https://galgos.inf.puc-rio.br")
```

## Arguments

| | |
|---|---|
| `qualifier` | The qualifier of the problem instance. E.g. "tai150d". This can either be inferred directly from the website or by the output of `cvrplib_ls()`. Can be a vector to download several instances at once, which is preferable to repeated calls since the list of available instances is only requested once. |
| `cvrplib_url` | The base URL of the CVRPLIB website. Defaults to https://galgos.inf.puc-rio.br. |

## Value

For a single qualifier, returns a "`cvrplib_instance`" object. For several qualifiers, returns a `list()` of those objects, named by qualifier. Note that a vector of length one counts as a single qualifier.

A "`cvrplib_instance`" object contains CVRPLIB problem instance data. Its `best_known_cost` is the cost of the best known solution provided by CVRPLIB (not necessarily a proven optimum), or `NA` if CVRPLIB provides no solution file for the instance. A solution file that exists but can not be downloaded raises an error.

Instances defined by an explicit distance matrix store it as `distances`, a `dist` object, and have `NA` coordinates in `pos`. Instances defined by coordinates have `distances` set to `NULL`.

## See Also

Other cvrplib: `clarke_wright_cvrplib()`, `cvrplib_ls()`

## cvrplib_E *CVRP instance data by Christofides and Eilon, 1969*

### Description

A collection of CVRP instances by Christofides and Eilon, 1969, provided courtesy of CVRPLIB. See CVRPLIB for visualizations of the instances and their solutions as well as a multitude of alternative instance data.

### Usage

```
cvrplib_E
```

### Format

`cvrplib_E`:
A list of CVRP instances as "`cvrplib_instance`" objects. The instances can be directly fed into solver algorithm, e.g. via `clarke_wright_cvrplib()`.

### Source

https://galgos.inf.puc-rio.br/cvrplib/en/instances

## cvrplib_F *CVRP instance data by Fisher, 1994*

### Description

A collection of CVRP instances by Fisher, 1994, provided courtesy of CVRPLIB. See CVRPLIB for visualizations of the instances and their solutions as well as a multitude of alternative instance data.

### Usage

```
cvrplib_F
```

### Format

`cvrplib_F`:
A list of CVRP instances as "`cvrplib_instance`" objects. The instances can be directly fed into solver algorithm, e.g. via `clarke_wright_cvrplib()`.

### Source

https://galgos.inf.puc-rio.br/cvrplib/en/instances

## cvrplib_ls *List available CVRPLIB online data*

### Description

Scrapes the CVRPLIB website to look for available data sets. This function call can take some time.

### Usage

```
cvrplib_ls(cvrplib_url = "https://galgos.inf.puc-rio.br")
```

### Arguments

| | |
|---|---|
| cvrplib_url | The base URL of the CVRPLIB website. Defaults to https://galgos.inf.puc-rio.br. |

### Value

A `data.frame()` where each row represents a CVRPLIB instance with the following columns:

- `name` - Instance qualifiers which can be used with `cvrplib_download()`.
- `instance_url` - The URL to the instance file.
- `solution_url` - The URL to the solution file, `NA` if there is none.

### See Also

Other cvrplib: `clarke_wright_cvrplib()`, `cvrplib_download()`

## cvrplib_Tai *CVRP instance data by Rochat and Taillard, 1995*

### Description

A collection of CVRP instances by Rochat and Taillard, 1995, provided courtesy of CVRPLIB. See CVRPLIB for visualizations of the instances and their solutions as well as a multitude of alternative instance data.

### Usage

```
cvrplib_Tai
```

### Format

`cvrplib_Tai`:
A list of CVRP instances as `"cvrplib_instance"` objects. The instances can be directly fed into solver algorithm, e.g. via `clarke_wright_cvrplib()`.

## Source

https://galgos.inf.puc-rio.br/cvrplib/en/instances

---

`milkr_cost` *Vehicle runs cost / distance*

---

## Description

Calculates the total distance associated to a `clarke_wright()` result. This is the measure that the corresponding Capacitated Vehicle Routing Problem minimizes.

## Usage

```
milkr_cost(solution)
```

## Arguments

| | |
|---|---|
| `solution` | A "`heumilkr_solution`" object, typically obtained by `clarke_wright()`. |

## Value

The total traveled distance.

## Examples

```
demand <- c(3, 2, 4, 2)

positions <-
  data.frame(
    pos_x = c(0, 1, -1, 2, 3),
    pos_y = c(0, 1, 1, 2, 3)
  )

solution <- clarke_wright(
  demand,
  dist(positions),
  data.frame(n = NA_integer_, caps = 6)
)

milkr_cost(solution)
```

`milkr_saving` *Vehicle run saving*

## Description

Measures the saving that was achieved by the heuristic optimization algorithm `clarke_wright()` compared to the naive vehicle run assignment, i.e. one run per site, or as many runs as the solution uses for a site whose demand exceeds the vehicle capacities.

## Usage

```
milkr_saving(solution, relative = FALSE)
```

## Arguments

| | |
|---|---|
| `solution` | A "`heumilkr_solution`" object, typically obtained by `clarke_wright()`. |
| `relative` | Should the saving be given as dimensionful value (in units of distance as provided to `clarke_wright()`), or as percentage relative to the naive costs. Defaults to `FALSE`, i.e. a dimensionful value. |

## Value

The savings either as dimensionful value or as percentage relative to the naive costs, depending on `relative`.

## Examples

```
demand <- c(3, 2, 4, 2)

positions <-
  data.frame(
    pos_x = c(0, 1, -1, 2, 3),
    pos_y = c(0, 1, 1, 2, 3)
  )

solution <- clarke_wright(
  demand,
  dist(positions),
  data.frame(n = NA_integer_, caps = 6)
)

print(milkr_saving(solution))
print(milkr_saving(solution, relative = TRUE))
```

`plot.heumilkr_solution` *Plot a CVRP solution*

## Description

Represents the sites and runs on a 2D plane so that the distances between sites on the drawn 2D plane correspond to `distances` provided to the solver `clarke_wright()`.

The individual runs are distinguished by color. The site locations are marked with round circles while the depot is depicted as a square. Delivery and pickup sites are not distinguished. The line types (solid/dashed/...) are associated to different vehicle types.

## Usage

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

## Arguments

| | |
|---|---|
| `x` | A "`heumilkr_solution`" object, typically obtained by `clarke_wright()`. |
| `...` | Not used. |

## Details

Distance information between sites only determine site positions on a 2D plane up to rotations and translations: those are fixed arbitrarily.

# Index


∗ **cvrplib**
    clarke_wright_cvrplib, 5
    cvrplib_download, 7
    cvrplib_ls, 9
∗ **datasets**
    cvrplib_A, 6
    cvrplib_B, 6
    cvrplib_E, 8
    cvrplib_F, 8
    cvrplib_Tai, 9

autoplot.heumilkr_solution, 2

clarke_wright, 3
clarke_wright(), *2*, *5*, *10–12*
clarke_wright_cvrplib, 5
clarke_wright_cvrplib(), *6–9*
cvrplib_A, *5*, 6
cvrplib_B, 6
cvrplib_download, 7
cvrplib_download(), *5*, *9*
cvrplib_E, 8
cvrplib_F, 8
cvrplib_ls, 9
cvrplib_ls(), *5*, *7*
cvrplib_Tai, 9

data.frame(), *3*, *4*, *9*

list(), *4*, *7*

milkr_cost, 10
milkr_saving, 11

numeric(), *3*

plot.heumilkr_solution, 12

stats::dist(), *3*