Package 'cayleyR'

November 25, 2025

Type Package

Title Cayley Graph Analysis for Permutation Puzzles

Version 0.1.0

Description Implements algorithms for analyzing Cayley graphs of permutation groups, with a focus on the TopSpin puzzle and similar permutation-based combinatorial puzzles. Provides methods for cycle detection, state space exploration, and finding optimal operation sequences in permutation groups generated by shift and reverse operations.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports digest

Suggests knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/Zabis13/cayleyR

BugReports https://github.com/Zabis13/cayleyR/issues

NeedsCompilation no

Author Yuri Baramykov [aut, cre]

Maintainer Yuri Baramykov <lbsbmsu@mail.ru>

Repository CRAN

Date/Publication 2025-11-25 20:42:14 UTC

Contents

`apply_operations` *Apply Operation Sequence*

Description

Applies a sequence of operations to a permutation state. Operations can be specified as "L"/"1" (shift left), "R"/"2" (shift right), or "X"/"3" (reverse first k elements).

Usage

```
apply_operations(state, operations, k)
```

Arguments

`state`	Integer vector representing the initial permutation state
`operations`	Character vector of operations to apply sequentially
`k`	Integer, the parameter for reverse_prefix operations

Value

Integer vector representing the state after all operations

Examples

```
# Basic usage with numeric codes
apply_operations(1:5, c("1", "3"), k = 3)

# TopSpin puzzle example
start_state <- 1:20
operations <- c("1", "3", "2")  # Left, Reverse(4), Right
result <- apply_operations(start_state, operations, k = 4)
print(result)

# Using letter codes
apply_operations(1:5, c("L", "X", "R"), k = 3)
```

`find_best_random_combinations` *Find Best Random Operation Sequences*

Description

Generates random sequences of operations and evaluates their cycle lengths to find sequences that produce the longest cycles in the Cayley graph. Useful for discovering interesting operation sequences in permutation puzzles.

Usage

```
find_best_random_combinations(
  moves,
  combo_length,
  n_samples,
  n_top,
  start_state,
  k
)
```

Arguments

`moves`	Character vector of allowed operation symbols (e.g., c("1", "2", "3") or c("L", "R", "X"))
`combo_length`	Integer, length of each operation sequence to test
`n_samples`	Integer, number of random sequences to generate and test
`n_top`	Integer, number of top results to return (sorted by cycle length)
`start_state`	Integer vector, initial permutation state
`k`	Integer, parameter for reverse operations

Details

The returned data frame `reachable_states_df` has the following structure:

```
step | state           | operation
-----|-----------------|----------
1    | 1 2 3 4 5...    | 1         <- INITIAL, operation="1" (next operation)
2    | 2 3 4 5 6...    | 1         <- after applying op1 from row 1
3    | 3 4 5 6 7...    | 2         <- after applying op1 from row 2
4    | 4 5 6 7 8...    | 3         <- after applying op2 from row 3
5    | 5 6 7 8 9...    | 1         <- after applying op3 from row 4
...
20   | 20 1 2 3 4...   | 1         <- after applying all ops from rows 1-19
NA   | 1 2 3 4 5...    | NA        <- after applying op1 from row 20 = initial
```

Value

Data frame with columns:

`combination`	String representation of the operation sequence
`total_moves`	Cycle length for this sequence
`unique_states_count`	Number of unique states visited in the cycle

Examples

```
# Find top 10 sequences from 100 random samples
best <- find_best_random_combinations(
  moves = c("1", "2", "3"),
  combo_length = 10,
  n_samples = 100,
  n_top = 10,
  start_state = 1:10,
  k = 4
)
print(best)

# Quick search with letter codes
top5 <- find_best_random_combinations(
  moves = c("L", "R", "X"),
  combo_length = 5,
  n_samples = 100,
  n_top = 5,
  start_state = 1:10,
  k = 3
)
print(top5)
```

`get_reachable_states` *Find Cycle in Permutation Group*

Description

Explores the Cayley graph starting from an initial state and applying a sequence of operations repeatedly until returning to the start state. Returns detailed information about all visited states and the cycle structure.

Usage

```
get_reachable_states(start_state, allowed_positions, k, verbose = FALSE)
```

Arguments

start_state	Integer vector, the initial permutation state
allowed_positions	Character vector, sequence of operations to repeat
k	Integer, parameter for reverse operations
verbose	Logical; if TRUE, prints progress and cycle information messages (default FALSE)

Value

List containing:

states	List of all visited states
reachable_states_df	Data frame with states, operations, and step numbers
operations	Vector of operations applied
total_moves	Total number of moves in the cycle
unique_states_count	Number of unique states visited
cycle_info	Summary string with cycle statistics

Examples

```
# Simple example with letter codes
result <- get_reachable_states(1:20, c("L", "X"), k = 4)
writeLines(result$cycle_info)

# Example with numeric codes
n <- 20
k <- 4
start_state <- 1:n
allowed_positions <- c("1", "3", "2")
result <- get_reachable_states(start_state, allowed_positions, k)
writeLines(result$cycle_info)
head(result$reachable_states_df)
```

---

get_reachable_states_light

Find Cycle Length (Lightweight Version)

Description

Fast version of cycle detection that only returns cycle length and unique state count without storing all intermediate states. Useful for testing many operation sequences efficiently.

Usage

```
get_reachable_states_light(start_state, allowed_positions, k)
```

Arguments

`start_state`	Integer vector, the initial permutation state
`allowed_positions`	Character vector, sequence of operations to repeat
`k`	Integer, parameter for reverse operations

Value

List containing:

`total_moves`	Total number of moves to return to start state
`unique_states_count`	Number of unique states in the cycle

Examples

```
# Quick cycle length check
result <- get_reachable_states_light(1:20, c("L", "X", "L"), k = 4)
cat("Cycle length:", result$total_moves, "\n")
cat("Unique states:", result$unique_states_count, "\n")

# Compare multiple sequences
seq1 <- get_reachable_states_light(1:20, c("1", "3"), k = 4)
seq2 <- get_reachable_states_light(1:20, c("2", "3"), k = 4)
cat("Sequence 1 cycle:", seq1$total_moves, "\n")
cat("Sequence 2 cycle:", seq2$total_moves, "\n")
```

`reverse_prefix` *Reverse Prefix*

Description

Reverses the first k elements of the state vector, analogous to the turnstile operation in the TopSpin puzzle.

Usage

```
reverse_prefix(state, k)
```

Arguments

`state`	Integer vector representing the current permutation state
`k`	Integer, the number of leading elements to reverse

Value

Integer vector with the first k elements reversed

Examples

```
# Basic example
reverse_prefix(1:10, 4)  # Returns c(4, 3, 2, 1, 5, 6, 7, 8, 9, 10)

# With variables
n <- 10
k <- 4
start_state <- 1:n
demo <- reverse_prefix(start_state, k)
print(demo)
```

`shift_left` *Shift State Left*

Description

Performs a cyclic left shift on the state vector, moving the first element to the end.

Usage

```
shift_left(state)
```

Arguments

`state`	Integer vector representing the current permutation state

Value

Integer vector with elements shifted left by one position

Examples

```
# Basic shift operation
shift_left(1:5)

# With variables
start_state <- 1:20
result <- shift_left(start_state)
print(result)
```

shift_right *Shift State Right*

Description

Performs a cyclic right shift on the state vector, moving the last element to the beginning.

Usage

```
shift_right(state)
```

Arguments

state	Integer vector representing the current permutation state

Value

Integer vector with elements shifted right by one position

Examples

```
# Simple example
shift_right(1:5)

# With variable assignment
start_state <- 1:20
result <- shift_right(start_state)
print(result)
```

Index