Package 'Coreset'

September 17, 2026

Title Discrete Diversity, Dispersion, and Coverage Subset Selection

Version 1.0.0

Description Solves discrete location objectives on a distance matrix or Euclidean coordinate set. The Max-Min Diversity (MMDP / p-dispersion) objective, which maximizes the minimum pairwise distance within a selection of k items, is solved by farthest-first selection (Gonzalez 1985) <doi:10.1016/0304-3975(85)90224-5>; the DropAdd tabu-search heuristic (Porumbel, Hao & Glover 2011) <doi:10.1007/s10479-011-0898-z>, GRASP with path-relinking (Resende, Marti, Gallego & Duarte 2010) <doi:10.1016/j.cor.2008.05.011>, and an exact node-packing integer program (Sayyady & Fathi 2016) <doi:10.1016/j.ejor.2016.02.026>. The Max-Mean Dispersion objective, which selects a subset of unrestricted size maximising the sum of its pairwise distances divided by the number of selected elements, is solved by reinforcement-learning-guided tabu search (Nijimbere et al. 2020) <doi:10.3934/jimo.2020115>. The discrete k-centre (min-max covering / facility location) objective, which chooses k centres to minimise the largest distance from any point to its nearest centre, is solved via the CDSh heuristic (Garcia-Diaz et al. 2017 <doi:10.1007/s10732-017-9345-x>, 2019 <doi:10.1109/ACCESS.2019.2933875>), and an exact minimum-cover integer program. The maximum-entropy (maxdet) objective, which maximises the log-determinant of a similarity kernel built from the distances (Shewry & Wynn 1987 <doi:10.1080/02664768700000020>; the mode of a determinantal point process, Kulesza & Taskar 2012 <doi:10.1561/2200000044>), is solved by greedy pivoted-Cholesky selection and, for small instances, exact enumeration.

License GPL (>= 3)

Encoding UTF-8

Language en-GB

Depends R (>= 4.1)

Imports cli (>= 3.0.0), Rcpp, Rdpack (>= 0.7), stats

RdMacros Rdpack

Suggests highs, knitr, Matrix, quarto, rprojroot, spelling, testthat (>= 3.0.0)

VignetteBuilder quarto

LinkingTo Rcpp

URL https://ms609.github.io/Coreset/

BugReports https://github.com/ms609/Coreset/issues

Config/Needs/benchmark bench, highs

Config/Needs/check rcmdcheck, testthat

Config/Needs/coverage covr

Config/Needs/memcheck testthat, pkgdown

Config/Needs/metadata codemeta

Config/Needs/revdeps revdepcheck

Config/Needs/website pkgdown

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/testthat/parallel false

ByteCompile true

NeedsCompilation yes

Author Martin R. Smith [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-5660-1727>)

Maintainer Martin R. Smith <martin.smith@durham.ac.uk>

Repository CRAN

Date/Publication 2026-09-17 14:50:29 UTC

Contents

DropAdd . . . 3
ExactKCentre . . . 5
ExactMaxMin . . . 6
ExactMaxSum . . . 8
FarFirst . . . 9
Grasp . . . 11
KCentre . . . 13
KCentreRadius . . . 14
MaxEntropy . . . 15
MaxMean . . . 17
MeanDist . . . 18
MinDist . . . 19
PickPoint . . . 20
print.Coreset . . . 21
print.KCentre . . . 22
print.MaxEntropy . . . 23
print.MaxMeanSelection . . . 24

print.MaxSum 25
summary.Coreset 26

Index **27**

`DropAdd` *DropAdd Tabu Search for the Max-Min Diversity Problem*

Description

`DropAdd()` selects a maximally-dispersed subset of `k` points using the DropAdd tabu search algorithm, which comprises a greedy construction followed by a first-in, first-out drop-add tabu search, with streamlined neighbour-evaluation tricks (algorithms 1–4 in Porumbel et al. 2011).

Usage

```
DropAdd(
  k,
  d = NULL,
  plateau = 5000L,
  maxSeconds = Inf,
  points = NULL,
  maxCandidates = 46340L,
  seed = NULL,
  N = NULL
)
```

Arguments

`k`	Integer: subset size, $2 \leq k \leq N$.
`d`	A `dist` object, a square symmetric numeric matrix, or a distance-column function (see below).
`plateau`	Integer: stop after this many consecutive drop-add iterations do not improve the score.
`maxSeconds`	Numeric: terminate search after this many seconds have elapsed.
`points`	A numeric $N \times \dim$ coordinate matrix (or an object coercible to one via `as.matrix`). Must be complete (no `NA`). Ignored if `d` specified. Avoids creating an $N \times N$ distance matrix, enabling use at $N \geq 46340$).
`maxCandidates`	Integer: when the number of candidate points N exceeds `maxCandidates`, the solver runs on a coreset of `maxCandidates` chosen by `FarFirst()`. `maxCandidates = 0` (or `Inf`) disables thinning.
`seed`	Optional integer specifying the index of an element with which to seed the warm-start search.
`N`	Integer: the total number of elements. Required only if `d` is a function.

Value

`DropAdd()` returns an integer vector of length `k` containing the selected indices, sorted ascending, with attributes:

score numeric specifying the achieved MaxMin objective $\min_{i \neq j \in S} d_{ij}$.

secondary numeric specifying the achieved (upper triangle) sum of pairwise distances over S.

seconds numeric specifying wall-clock seconds spent.

iters integer specifying main-loop iterations executed, excluding the construction phase.

The vector has class `"MaxMinSelection"` and prints as a one-line summary (see `print.MaxMinSelection()`).

Progress bar

In interactive sessions, status messages are shown. To toggle, set `options("Coreset.progress" = FALSE)` (or `TRUE`).

Parallelism

To parallelize computation when OpenMP is available, set the `"mc.cores"` option:

```
options(mc.cores = 2L)                        # use a fixed number of cores
options(mc.cores = parallel::detectCores())  # or all available cores
```

Distance function

When `d` is a function, `d(i)` must return the distances from element `i` to every element (length `N`, with the self-distance ignored), or to every element except `i` (length `N - 1`, in order). `N` is required, and memory is $O(N)$. This suits metrics where no stored matrix or coordinate embedding is available.

Because `d` is typically called many times; specifying a distance matrix (where memory permits) is likely to require less calculation than multiple calls to `d`, unless `d` implements efficient caching.

References

Porumbel D, Hao J, Glover F (2011). "A simple and effective algorithm for the MaxMin diversity problem." *Annals of Operations Research*, **186**, 275–293. doi:10.1007/s104790110898z.

Examples

```
set.seed(1)
pts <- matrix(rnorm(200), ncol = 2)
DropAdd(5L, dist(pts))

# Composable coreset: thin to 40 candidates with farthest-first, then run
# DropAdd on the coreset. Returned indices are original-space row indices.
suppressWarnings(DropAdd(5L, points = pts, maxCandidates = 40L))

# Disable thinning on the full problem
DropAdd(5L, points = pts, maxCandidates = 0L)
```

```
# Distance function; `cache` memoizes the columns to reduce computation.
data("USArrests")
ArrestDist <- function(dat) {
  scaled <- scale(as.matrix(dat))                # derived once
  cache <- new.env(parent = emptyenv())
  function(i) {
    key <- as.character(i)
    if (is.null(cache[[key]])) {
      cache[[key]] <- sqrt(rowSums(sweep(scaled, 2, scaled[i, ], "-") ^ 2))
    }
    cache[[key]]
  }
}
arrests <- USArrests[, c("Murder", "Assault", "Rape")]
idx <- DropAdd(4L, ArrestDist(arrests), N = nrow(arrests), plateau = 200L)
USArrests[idx, ]
```

`ExactKCentre` *Exact discrete* k*-centre optimum*

Description

`ExactKCentre()` finds an optimal solution to the discrete *k*-centre problem.

Usage

```
ExactKCentre(k, d, maxSeconds = 60)

ExactKCenter(k, d, maxSeconds = 60)
```

Arguments

`k`	Integer specifying maximum number of centres to identify, from 1 to `nrow(d)`.
`d`	`dist` object or a square symmetric numeric distance matrix.
`maxSeconds`	Numeric specifying wall-clock budget, in seconds, for the search. If the time expires before the optimum is proven, the smallest radius proven feasible is returned, with the attribute `proven = FALSE`.

Details

The optimum covering radius is the smallest threshold `r`, over the achieved distinct distances, for which `k` centres can cover every point within `r`.

Each probe asks whether `k` centres cover every point within a candidate radius. This is decided combinatorially via unit propagation and dominance reduction, then an exhaustive component-wise search. The search is warm-started from the `KCentre()` radius, then bisects downwards.

Value

`ExactKCentre()` returns an integer vector of length $\leq k$ listing the chosen centres in ascending order. It has class `c("KCentreExact", "KCentreSelection")` and attributes:

radius The covering radius achieved; the proven optimum when `proven` is `TRUE`, otherwise an upper bound.

proven Logical: `TRUE` if optimality is certified.

seconds Wall-clock seconds elapsed.

N, k Instance size and centre budget.

Progress bar

In interactive sessions, a progress indicator is shown. To toggle, set `options("Coreset.progress" = FALSE)` (or `TRUE`).

See Also

`KCentre()` for the fast near-optimal heuristic; `ExactMaxMin()` for the dual MMDP optimum.

Examples

```
set.seed(1)
pts <- matrix(rnorm(40), ncol = 2)
d <- dist(pts)
ExactKCentre(3L, d)
```

ExactMaxMin *Exact Max-Min Diversity Problem solution*

Description

`ExactMaxMin()` finds the optimal solution to the Max-Min Diversity Problem (discrete p-dispersion) by iterated node-packing (Sayyady and Fathi 2016) (which may be slow or intractable on large sets).

Usage

```
ExactMaxMin(
  k,
  d,
  maxSeconds = 60,
  warmStart = NULL,
  nStart = 1L,
  graspPlateau = 50L,
  dropPlateau = 512L
)
```

Arguments

k	Integer: target subset size, between 2 and nrow(d).
d	dist object or a square symmetric numeric distance matrix.
maxSeconds	Numeric: search terminates after this many seconds have elapsed, returning largest threshold proven feasible.
warmStart	Optional integer vector giving indices of a candidate subset to add to the heuristic warm-start pool.
nStart	Integer: how many Grasp() restarts enter the warm-start pool.
graspPlateau, dropPlateau	Integer: the stopping plateaus given to the pool's Grasp() restarts and its DropAdd() pass. Deeper searches cost more, but raise the lower bound the exact search starts from.

Details

The search is warm-started from a heuristic lower bound (the best of nStart Grasp() restarts and a DropAdd() pass), then gallops upward from that bound to the first infeasible threshold and bisects the resulting bracket.

To parallelize computation when OpenMP is available, set the "mc.cores" option:

```
options(mc.cores = 2L)                       # use a fixed number of cores
options(mc.cores = parallel::detectCores())  # or all available cores
```

Parallelization returns identical results under a given seed.

Value

ExactMaxMin() returns an integer vector of length k (sorted ascending) with class "MaxMinSelection", carrying attributes:

score The minimum pairwise distance within the selection. When proven is TRUE this is the optimum; otherwise a lower bound.

proven Logical: TRUE if the search certified optimality within the budget, FALSE if it returned an unproven incumbent.

seconds Wall-clock seconds elapsed.

N, k Instance size and target subset size.

Prints as a terse summary via print.MaxMinSelection().

Progress bar

In interactive sessions, a progress indicator is shown. To toggle, set options("Coreset.progress" = FALSE) (or TRUE).

References

Sayyady F, Fathi Y (2016). "An integer programming approach for solving the p-dispersion problem." *European Journal of Operational Research*, **253**(1), 216–225. doi:10.1016/j.ejor.2016.02.026.

Examples

```
set.seed(1)
pts <- matrix(rnorm(18), ncol = 2)
ExactMaxMin(3L, dist(pts))
```

`ExactMaxSum` *Exact Maximum Diversity Problem (max-sum) solution*

Description

`ExactMaxSum()` finds the optimal solution to the Max-Sum Diversity Problem (the "maximum diversity problem"): it selects the `k` points that maximizes the total pairwise distance between points. As the problem is NP-hard, it is feasible only for small sets.

Usage

```
ExactMaxSum(k, d, maxSeconds = 30, warmStart = NULL)
```

Arguments

`k`	Integer: target subset size, between 2 and `nrow(d)`.
`d`	`dist` object or a square symmetric numeric distance matrix.
`maxSeconds`	Numeric: search terminates after this many seconds have elapsed, returning largest threshold proven feasible.
`warmStart`	Optional integer vector giving indices of a candidate subset to add to the heuristic warm-start pool.

Details

The solver uses per-node integer-program linearisation (Kuo et al. 1993), starting from a multi-start 1-swap local search, whose result is returned when optimality cannot be proven within `maxSeconds`.

Value

`ExactMaxSum()` returns an integer vector of length `k`, sorted ascending, with class `"MaxSumSelection"`, carrying attributes:

score Numeric specifying the achieved total pairwise distance within the selection. When `proven` is `TRUE` this is the optimum; otherwise a lower bound.

proven Logical: `TRUE` if optimality was certified.

seconds, N, k Numerics reporting the wall-clock seconds elapsed; instance size; and target size.

References

Kuo C, Glover F, Dhir KS (1993). "Analyzing and modeling the maximum diversity problem by zero-one programming." *Decision Sciences*, **24**(6), 1171–1185. doi:10.1111/j.15405915.1993.tb00509.x.

Examples

```
set.seed(1)
pts <- matrix(rnorm(20), ncol = 2)
# Package 'highs' is required for ExactMaxSum
if (requireNamespace("highs", quietly = TRUE)) {
  ExactMaxSum(3L, dist(pts))
}
```

`FarFirst` *Greedy farthest-first point selection*

Description

Greedy farthest-first selection (González 1985; Hochbaum and Shmoys 1985) iteratively selects the point furthest from the current selection to yield a 2-approximation to the *k*-centre and Max Min Diversity problems.

Usage

```
FarFirst(
  k,
  d = NULL,
  points = NULL,
  N = NULL,
  strategy = "random_furthest",
  nSeeds = 3L
)
```

Arguments

`k`	Integer: number of points to select.
`d`	A `dist` object, a square numeric matrix of pairwise distances, or a distance function that takes an index `i` and returns the distance from `i` to each other element (optionally including the self-distance). Ignored when `points` is supplied.
`points`	Optional `N x dim` numeric coordinate matrix. When supplied, the selection is computed directly from coordinates in $O(N \cdot k \cdot dim)$ time and $O(N)$ memory, which avoids creating an $O(N^2)$ distance matrix. Missing entries (`NA`) are not supported.
`N`	Integer: the total number of elements. Required (and used) only on the distance-column oracle path, where it cannot be inferred from the closure; ignored for the matrix and coordinate paths.
`strategy`	Integer or character defining how to seed the greedy pass. Pass the name of one or more seeding strategies described in `PickPoint()` to run each strategy and return the best solution.
`nSeeds`	Integer: number of distinct seeds to draw under the (default) `"random_furthest"` strategy. Beyond ~3, `DropAdd()` will tend to return higher quality results faster.

Value

`FarFirst()` returns an integer vector with class `MaxMinSelection`, listing the selected indices in the order they were selected. Attributes report:

- `score`: the selection's minimum pairwise distance (T_k).
- `winning_strategy`: character vector listing strategies that attained the optimal score.
- `strategy_results`: results for each strategy.

Progress bar

In interactive sessions, the distance-column path shows a progress bar. To toggle, set `options("Coreset.progress" = FALSE)` (or `TRUE`).

Parallelism

To parallelize computation when OpenMP is available, set the `"mc.cores"` option:

```
options(mc.cores = 2L)                       # use a fixed number of cores
options(mc.cores = parallel::detectCores())  # or all available cores
```

The main use case for parallelization is when `nSeeds` is a multiple of `mc.cores`, so each seed point can be evaluated in parallel.

References

González TF (1985). "Clustering to minimize the maximum intercluster distance." *Theoretical Computer Science*, **38**, 293–306. doi:10.1016/03043975(85)902245.

Hochbaum DS, Shmoys DB (1985). "A best possible heuristic for the k-center problem." *Mathematics of Operations Research*, **10**(2), 180–184. doi:10.1287/moor.10.2.180.

See Also

`PickPoint()` for the seed indices alone; `DropAdd()` and `ExactMaxMin()` for higher-effort solvers.

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
d <- dist(pts)

# Default: best of three random-furthest starts (set.seed for reproducibility):
FarFirst(5L, d)

# More random-furthest starts:
FarFirst(5L, d, nSeeds = 15L)

# Custom two-anchor ensemble:
FarFirst(5L, d, strategy = c("diameter", "anti_medoid"))
```

```
# A single strategy:
FarFirst(5L, d, strategy = "diameter")

# An explicit start index (integer strategy):
FarFirst(5L, d, strategy = 1L)

# Matrix-free coordinate path (identical result, O(N) memory):
FarFirst(5L, points = pts, strategy = 1L)

# Distance-column oracle: supply one column at a time, never the full matrix.
data("USArrests")
arrestTypes <- USArrests[, c("Murder", "Assault", "Rape")]
StateDist <- function(i) {
  diffs <- sweep(arrestTypes, 2, unlist(arrestTypes[i, ]), "-")
  sqrt(rowSums(diffs ^ 2))
}
idx <- FarFirst(4L, StateDist, N = nrow(arrestTypes), strategy = 1L)
arrestTypes[idx, ]
```

`Grasp` *GRASP with Path Relinking for the Max-Min Diversity Problem*

Description

`Grasp()` solves the Max-Min Diversity Problem (discrete p-dispersion) with the static variant of the GRASP / path-relinking metaheuristic (Resende et al. 2010, fig. 4). This expensive heuristic often attains high-quality selections.

Usage

```
Grasp(
  k,
  d,
  plateau = 100L,
  eliteSize = 10L,
  alpha = 0.8,
  maxSeconds = Inf,
  maxCandidates = 2000L
)
```

Arguments

`k`	Integer subset size, `2 <= k <= nrow(d)`.
`d`	Either a `dist` object or a square symmetric numeric matrix.
`plateau`	Integer; stop after this many consecutive GRASP iterations have not improved the best elite objective.
`eliteSize`	Size of the elite set $\|ES\|$.

`alpha`	Numeric in `[0, 1]` specifying construction greediness. Each step draws the next point at random from a shortlist of the strongest candidates whose gain lies within a fraction `alpha` of the best-to-worst spread. `alpha = 1` is pure greedy; `alpha = 0` is uniform random among candidates.
`maxSeconds`	Numeric specifying wall-clock ceiling, in seconds.
`maxCandidates`	Integer: when the number of candidate points N exceeds `maxCandidates`, the solver runs on a coreset of `maxCandidates` chosen by `FarFirst()`. `maxCandidates = 0` (or `Inf`) disables thinning.

Details

The GRASP with path-relinking algorithm conducts a randomised-greedy construction with extended-improvement local search builds; it identifies an elite set, then conducts a single pass of path relinking over all elite pairs (Resende et al. 2010).

The refinement loop stops after `plateau` consecutive GRASP iterations fail to improve the best elite objective, or once `maxSeconds` have elapsed.

This method will fail if the complete $N \times N$ distance matrix is too large to fit into memory.

Value

`Grasp()` returns an integer vector of length `k` specifying the indices of the selected points, with attributes:

score Achieved MaxMin objective T_k.

seconds Wall-clock seconds spent.

iters Number of GRASP refinement iterations executed.

pr_calls Number of path-relinking pair-applications run.

The vector has class `"MaxMinSelection"` and prints as a one-line summary (see `print.MaxMinSelection()`).

Parallelism

To parallelize computation when OpenMP is available, set the `"mc.cores"` option:

```
options(mc.cores = 2L)                       # use a fixed number of cores
options(mc.cores = parallel::detectCores())  # or all available cores
```

Progress bar

In interactive sessions, a bar tracks how close the search is to its `plateau` stopping criterion, snapping back each time a better solution is found. To toggle, set `options("Coreset.progress" = FALSE)` (or `TRUE`).

References

Resende MGC, Martí R, Gallego M, Duarte A (2010). "GRASP and path relinking for the max-min diversity problem." *Computers & Operations Research*, **37**(3), 498–508. doi:10.1016/j.cor.2008.05.011.

See Also

DropAdd() for scalable refinement; ExactMaxMin() for the proven optimum on small instances.

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
Grasp(5L, dist(pts), plateau = 20L, eliteSize = 4L)

# Composable coreset: thin to 20 candidates with farthest-first, then run
# GRASP on the coreset. Returned indices are original-space row indices.
suppressWarnings(Grasp(5L, dist(pts), plateau = 20L, maxCandidates = 20L))
```

KCentre *Discrete* k*-centre solver*

Description

KCentre() selects k elements (centres) so as to minimize the largest distance from any point to its nearest centre (the covering radius), using the Critical Dominating Set heuristic (CDSH) (García-Díaz et al. 2017; García-Díaz et al. 2019).

Usage

```
KCentre(k, d, nstart = 1L, effort = 1L)

KCenter(k, d, nstart = 1L, effort = 1L)
```

Arguments

k	Integer specifying maximum number of centres to identify, from 1 to nrow(d).
d	dist object or a square symmetric numeric distance matrix.
nstart	Integer specifying how many deterministic peripheral seeds to try.
effort	Integer: if > 0, run effort parallel FarFirst() searches, each seeded on a distinct starting point, and return the best result found.

Details

On the benchmark instances of García-Díaz et al. (2019), the CDS heuristic reaches roughly 1–3.5% of the optimum at $O(N^2 \log N)$, far tighter than FarFirst().

Despite this good performance in practice, the CDSH is a 3-approximation. To guard against occasional cases where a better candidate is missed, KCentre() runs by default an exhaustive search of a small candidate grid (for n up to ~150); and an additional FarFirst() pass (controlled via the effort argument). These safeguards ensure that KCentre() always returns at least a 2-approximation.

Value

`KCentre()` returns an integer vector of length $\leq k$ specifying the chosen centres in ascending order. The achieved covering radius is attached as attribute `radius`. The vector has class `"KCentreSelection"` and prints as a one-line summary.

References

García-Díaz J, Menchaca-Méndez R, Menchaca-Méndez R, Pomares Hernández S, Pérez-Sansalvador JC, Lakouari N (2019). "Approximation algorithms for the vertex k-center problem: survey and experimental evaluation." *IEEE Access*, **7**, 109228–109245. doi:10.1109/ACCESS.2019.2933875.

García-Díaz J, Sánchez-Hernández J, Menchaca-Méndez R, Menchaca-Méndez R (2017). "When a worse approximation factor gives better performance: a 3-approximation algorithm for the vertex k-center problem." *Journal of Heuristics*, **23**(5), 349–366. doi:10.1007/s107320179345x.

González TF (1985). "Clustering to minimize the maximum intercluster distance." *Theoretical Computer Science*, **38**, 293–306. doi:10.1016/03043975(85)902245.

See Also

`ExactKCentre()` for the proven optimum; `KCentreRadius()` for a selection's score; `FarFirst()` for the González (1985) 2-approximation baseline.

Examples

```
set.seed(1)
pts <- matrix(rnorm(120), ncol = 2)
d <- dist(pts)
centres <- KCentre(5L, d)
KCentreRadius(d, centres)
# Results will beat a Gonzalez 2-approximation:
KCentreRadius(d, FarFirst(5L, d, nSeeds = 1))
```

`KCentreRadius` *Covering radius of a set of centres*

Description

`KCentreRadius()` computes the covering radius of a set of centres: the largest distance from any of the `N` points to its nearest centre, $R = \max_p \min_{c \in \mathrm{idx}} d(p, c)$. This is the min-max k-centre objective (González 1985) minimized by `KCentre()` and `ExactKCentre()`.

Usage

```
KCentreRadius(d = NULL, idx, points = NULL)

KCenterRadius(d = NULL, idx, points = NULL)
```

Arguments

d	Pairwise distance matrix or dist object. Ignored when points is supplied.
idx	Integer vector of centre indices (>= 1).
points	N x dim numeric coordinate matrix. When supplied, the per-point nearest-centre distances are computed from coordinates one column at a time. This supports sets with large N, in which the full N x N matrix would overflow available memory.

Value

KCentreRadius() returns a numeric denoting the covering radius.

References

González TF (1985). "Clustering to minimize the maximum intercluster distance." *Theoretical Computer Science*, **38**, 293–306. doi:10.1016/03043975(85)902245.

See Also

KCentre() and ExactKCentre() (which minimise this); MinDist() for the complementary MMDP objective.

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
d <- dist(pts)
centres <- KCentre(4L, d)
KCentreRadius(d, centres)
```

MaxEntropy *Maximum-entropy (maxdet) subset selection*

Description

MaxEntropy() selects the k points that maximise the log-determinant of their kernel block, $\log\det K_S$. This is equivalent to finding the set of k points that span the largest volume, which corresponds to the maximum-entropy sampling criterion (Shewry and Wynn 1987) and the maximum-*a-posteriori* mode of a determinantal point process (Kulesza and Taskar 2012).

Usage

```
MaxEntropy(
  k,
  d,
  sigma = NULL,
  repair = c("clip", "shift", "truncate"),
```

```
  exact = NA,
  maxCombos = 300000L
)
```

Arguments

k	Integer specifying target selection size, $1 \leq k \leq n$.
d	`dist` object or square numeric distance matrix over the n points.
sigma	Optional numeric specifying kernel bandwidth; defaults to the median positive distance.
repair	Character selecting a positive semi-definite repair method: `"clip"` (nearest), `"shift"` (diagonal loading) or `"truncate"` (low-rank embedding).
exact	Logical: `TRUE` uses explicit enumeration, failing with an error if `maxCombos` is exceeded; `FALSE` uses the greedy approximation. `NA` uses exact enumeration when `choose(n, k) <= maxCombos`, greedy otherwise.
maxCombos	Integer specifying ceiling on `choose(n, k)` for exact enumeration.

Details

A radial-basis kernel $K_{ij} = \exp(-d_{ij}^2/2\sigma^2)$ is built from the supplied distances and repaired to a positive-semidefinite matrix. The exact argmax is NP-hard (Kulesza and Taskar 2012). A greedy approximation is built by pivoted Cholesky, adding at each step the point of largest residual conditional variance. Ties are broken by selecting the more peripheral point.

Value

`MaxEntropy()` returns an integer vector of length `k` (sorted ascending) with class `"MaxEntropySelection"`, carrying attributes:

score The retained $\log\det K_S$ of the selection. `-Inf` is returned for a degenerate selection where `k` exceeds the number of distinct points.

negMass Fraction of spectral mass removed by the positive semi-definite repair.

sigma, repair, exact The bandwidth, repair, and whether the optimum was certified by enumeration.

seed, N, k The peripheral seed index, instance size, target size.

References

Kulesza A, Taskar B (2012). "Determinantal point processes for machine learning." *Foundations and Trends in Machine Learning*, **5**(2–3), 123–286. doi:10.1561/2200000044.

Shewry MC, Wynn HP (1987). "Maximum entropy sampling." *Journal of Applied Statistics*, **14**(2), 165–170. doi:10.1080/02664768700000020.

Examples

```
set.seed(1)
pts <- matrix(rnorm(40), ncol = 2)
MaxEntropy(4L, dist(pts))
```

MaxMean *Max-Mean Dispersion Problem solver*

Description

`MaxMean()` selects a maximally dispersed subset of elements from a pairwise distance matrix, maximising the *max-mean* objective:

$$f(S) = \frac{\sum_{i<j,\, i,j \in S} d_{ij}}{|S|}$$

The number of elements in the subset $|S| \geq 2$ is chosen so as to maximise the mean dispersion.

Usage

```
MaxMean(d, maxSeconds = 0.1, maxIter = 1000, useRL = TRUE)
```

Arguments

`d`	A `dist` object or square numeric matrix of pairwise distances; values may be negative, and an asymmetric matrix is symmetrized to $(d_{ij} + d_{ji})/2$ before solving.
`maxSeconds`	Numeric: wall-clock time budget, in seconds.
`maxIter`	Numeric: cap on the total tabu-search iterations across restarts.
`useRL`	Logical: if `TRUE` (the default) a Q-learning layer guides initial-solution construction across restarts; if `FALSE`, each restart starts from a random solution.

Details

`MaxMean()` implements the reinforcement-learning tabu search algorithm of (Nijimbere et al. 2020). An initial solution is constructed randomly for the first restart and via Q-learning thereafter; each initial solution is then refined by a tabu search using one-flip moves (adding or removing one element per step). Restarts continue until either the `maxSeconds` or `maxIter` budget is reached.

The reinforcement-learning and tabu hyperparameters are fixed at the tuned values reported by Nijimbere et al. (2020): greedy factor $\epsilon = 0.7$, learning rate $\alpha = 0.5$, discount $\gamma = 0.5$, maximum tabu tenure 120, search depth 50 000.

Value

`MaxMean()` returns an integer vector of selected 1-based indices (sorted ascending) with class `"MaxMeanSelection"` and attributes:

score numeric, achieved objective $\sum_{i<j \in S} d_{ij}/|S|$.

size integer, number of selected elements $|S|$.

seconds numeric, wall-clock seconds spent.

iters numeric, total tabu-search iterations across restarts.

The vector has class "MaxMeanSelection" and prints as a one-line summary (see print.MaxMeanSelection()); it is otherwise an ordinary integer vector that indexes the distance matrix directly.

Progress bar

In interactive sessions, status messages are shown. To toggle, set options("Coreset.progress" = FALSE) (or TRUE).

References

Nijimbere D, Zhao S, Gu X, Esangbedo MO, Dominique N (2020). "Tabu search guided by reinforcement learning for the max-mean dispersion problem." *Journal of Industrial & Management Optimization*, **17**, 3223–3254. doi:10.3934/jimo.2020115.

See Also

MeanDist() to score an arbitrary selection under this objective; FarFirst(), DropAdd() and Grasp() for fixed-cardinality max-min solvers.

Examples

```
# The max-mean problem is defined for signed dissimilarities; with these the
# optimal subset has an interior size, chosen to maximise mean dispersion.
set.seed(1)
x <- matrix(runif(100, -5, 5), 10)
d <- (x + t(x)) / 2          # symmetric, signed
selection <- MaxMean(d)
selection                    # 5 of the 10 elements: {2, 5, 6, 8, 10}
MeanDist(d, selection)       # 2.714 — equals attr(selection, "score")
```

MeanDist *Mean dispersion of a selection*

Description

MeanDist() reports the sum of pairwise distances divided by the number of selected elements,

$$f(S) = \frac{\sum_{i<j,\, i,j \in S} d_{ij}}{|S|}$$

, the objective maximised by MaxMean().

Usage

```
MeanDist(d, idx)
```

Arguments

d	Pairwise distance matrix or dist object.
idx	Integer vector of selected row/col indices.

Value

MeanDist() returns a numeric scalar, or NA_real_ if length(idx) < 2.

See Also

MaxMean() which maximises this objective; MinDist() for the max-min (MMDP) analogue.

Examples

```
# The max-mean problem is defined for signed dissimilarities; with these the
# optimal subset has an interior size, chosen to maximise mean dispersion.
set.seed(1)
x <- matrix(runif(100, -5, 5), 10)
d <- (x + t(x)) / 2          # symmetric, signed
selection <- MaxMean(d)
selection                    # 5 of the 10 elements: {2, 5, 6, 8, 10}
MeanDist(d, selection)       # 2.714 — equals attr(selection, "score")
```

MinDist *Minimum pairwise distance within a selection*

Description

Returns the minimum pairwise distance among selected points T_k. A set of points that is more dispersed will exhibit a higher value.

Usage

```
MinDist(d = NULL, idx, points = NULL)
```

Arguments

d	Pairwise distance matrix or dist object. Ignored when points is supplied.
idx	Integer vector of selected row/col indices.
points	Optional N x dim numeric coordinate matrix. When supplied, the score is computed from stats::dist() on the selected sub-coordinates only (k x k).

Details

The solvers in this package (FarFirst(), DropAdd(), Grasp()) already attach the achieved T_k as a score attribute. MinDist() allows arbitrary selections to be scored, or an existing selection to be scored against a different distance matrix.

Value

`MinDist()` returns a numeric specifying the minimum distance between two selected points (or `NA_real_`, if `length(idx) < 2`).

See Also

`FarFirst()`, `DropAdd()`, `Grasp()` and `ExactMaxMin()`.

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
d <- dist(pts)
MinDist(d, FarFirst(5L, d))
```

PickPoint *Seed to initialize farthest-first selection*

Description

`PickPoint()` implements a range of strategies to select a seed for greedy farthest-first selection. Propitious seeds yield better solutions.

Usage

```
PickPoint(
  d = NULL,
  points = NULL,
  strategy = c("peripheral", "anti_centroid", "random_furthest", "diameter",
    "anti_medoid", "medoid", "rowsum", "rownorm")
)
```

Arguments

`d`	A `dist` object or square symmetric numeric matrix. Ignored when `points` is supplied.
`points`	Optional `N x dim` numeric coordinate matrix; when supplied the seed is computed from coordinates in `O(N)` memory. Required for the `"anti_centroid"` anchor, which has no distance-matrix form.
`strategy`	Character specifying method to employ:

`"peripheral"` **(default)** Two sweeps: the point furthest from point 1, then the point furthest from that (a diameter-endpoint approximation). $O(N)$.

`"anti_centroid"` The point farthest from the coordinate mean ($\arg\max \|x - \bar{x}\|$). $O(N * dim)$. Requires `points`.

`"random_furthest"` The point furthest from a random pivot. $O(N)$.

`"diameter"` A row endpoint of the diameter pair (the maximum pairwise distance).

`"medoid"` The 1-median (medoid): the point minimising the sum of distances to all others.

`"anti_medoid"` The point furthest from the 1-median (medoid).

`"rowsum"` The point maximising the sum of distances to all others (the 1-anti-median).

`"rownorm"` The point maximising $\sqrt{(\sum d^2)}$, the L2 counterpart of `"rowsum"`.

Value

`PickPoint()` returns an integer that identifies the index of a proposed seed in `d` or `points`.

See Also

`FarFirst()`, which seeds and runs the greedy pass in one call.

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
d <- dist(pts)
PickPoint(d, strategy = "diameter")
FarFirst(5L, d, strategy = PickPoint(d, strategy = "diameter"))
```

---

print.Coreset *Format and print Coreset solver results*

Description

Terse summaries of the objects returned by the Coreset solvers.

Usage

```
## S3 method for class 'MaxMinSelection'
format(x, ...)

## S3 method for class 'MaxMinSelection'
print(x, ...)
```

Arguments

`x`	A `MaxMinSelection` object (from any solver, including `ExactMaxMin()`).
`...`	Ignored; present for S3 compatibility.

Value

`print.Coreset()` returns `x`, invisibly. It is called for its side-effect of printing `format(x)` to the console. `format.Coreset()` returns a character string reporting the selection size, the selected indices, the algorithm (and if applicable strategy or proof status), and the achieved T_k.

See Also

Other reporting functions: `print.KCentre`, `print.MaxEntropy`, `print.MaxMeanSelection()`, `print.MaxSum`, `summary.Coreset`

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
print(FarFirst(5L, dist(pts)))
```

---

print.KCentre *Format and print k-centre solver results*

Description

Terse summaries of the objects returned by `KCentre()` (`"KCentreSelection"`) and `ExactKCentre()` (`"KCentreExact"`)

Usage

```
## S3 method for class 'KCentreSelection'
format(x, ...)

## S3 method for class 'KCentreSelection'
print(x, ...)

## S3 method for class 'KCentreExact'
format(x, ...)

## S3 method for class 'KCentreExact'
print(x, ...)
```

Arguments

`x`	A `"KCentreSelection"` or `"KCentreExact"` object.
`...`	Ignored; present for S3 compatibility.

Value

`print.KCentre()` returns `x`, invisibly. It is called for its side-effect of printing `format(x)` to the console. `format.KCentre()` returns a character string describing a `KCentreSelection`; it reports the centre count, the chosen indices, the method, and the achieved covering radius (with proof status for the exact solver).

See Also

Other reporting functions: `print.Coreset`, `print.MaxEntropy`, `print.MaxMeanSelection()`, `print.MaxSum`, `summary.Coreset`

Examples

```
set.seed(1)
KCentre(4L, dist(matrix(rnorm(60), ncol = 2)))
```

`print.MaxEntropy` *Format and print maximum-entropy (maxdet) solver results*

Description

Terse summary of the object returned by `MaxEntropy()` (`"MaxEntropySelection"`), reporting the retained log-determinant ($\log\det K_S$, the maxdet objective) and the magnitude of the positive-semidefinite repair.

Usage

```
## S3 method for class 'MaxEntropySelection'
format(x, ...)

## S3 method for class 'MaxEntropySelection'
print(x, ...)
```

Arguments

`x`	A `"MaxEntropySelection"` object.
`...`	Ignored; present for S3 compatibility.

Value

`format.MaxEntropySelection()` returns a one-line character summary; `print.MaxEntropySelection()` returns `x` invisibly, called for its side-effect.

See Also

Other reporting functions: `print.Coreset`, `print.KCentre`, `print.MaxMeanSelection()`, `print.MaxSum`, `summary.Coreset`

Examples

```
set.seed(1)
MaxEntropy(4L, dist(matrix(rnorm(40), ncol = 2)))
```

print.MaxMeanSelection *Format and print Max-Mean solver results*

Description

Terse one-line and detailed summaries of the objects returned by MaxMean().

Usage

```
## S3 method for class 'MaxMeanSelection'
format(x, ...)

## S3 method for class 'MaxMeanSelection'
print(x, ...)

## S3 method for class 'MaxMeanSelection'
summary(object, ...)
```

Arguments

x	A MaxMeanSelection object returned by MaxMean().
...	Ignored; present for S3 compatibility.
object	A MaxMeanSelection object returned by MaxMean().

Value

print.MaxMeanSelection() returns x, invisibly. format.MaxMeanSelection() returns a character string reporting the selection size, the selected indices, and the achieved max-mean objective $f(S)$.

See Also

Other reporting functions: print.Coreset, print.KCentre, print.MaxEntropy, print.MaxSum, summary.Coreset

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
print(MaxMean(dist(pts), maxSeconds = 1))
```

print.MaxSum *Format and print Max-Sum (maximum diversity) solver results*

Description

Terse summary of the object returned by ExactMaxSum() ("MaxSumSelection"), reporting the achieved **total** pairwise distance (the max-sum objective) rather than the minimum distance of ExactMaxMin().

Usage

```
## S3 method for class 'MaxSumSelection'
format(x, ...)

## S3 method for class 'MaxSumSelection'
print(x, ...)
```

Arguments

x	A "MaxSumSelection" object.
...	Ignored; present for S3 compatibility.

Value

format.MaxSumSelection() returns a one-line character summary; print.MaxSumSelection() returns x invisibly, called for its side-effect.

See Also

Other reporting functions: print.Coreset, print.KCentre, print.MaxEntropy, print.MaxMeanSelection(), summary.Coreset

Examples

```
set.seed(1)
# Package 'highs' is required for ExactMaxSum()
if (requireNamespace("highs", quietly = TRUE)) {
  ExactMaxSum(3L, dist(matrix(rnorm(20), ncol = 2)))
}
```

summary.Coreset *Detailed summaries of Coreset solver results*

Description

A fuller counterpart to print.MaxMinSelection(): the one-line headline, followed by the achieved objective(s), search effort, and – for a FarFirst() ensemble – the per-strategy T_k table.

Usage

```
## S3 method for class 'MaxMinSelection'
summary(object, ...)
```

Arguments

object	A MaxMinSelection object (from any solver).
...	Ignored; present for S3 compatibility.

Value

summary.Coreset() returns object, invisibly.

See Also

Other reporting functions: print.Coreset, print.KCentre, print.MaxEntropy, print.MaxMeanSelection(), print.MaxSum

Examples

```
set.seed(1)
pts <- matrix(rnorm(60), ncol = 2)
summary(FarFirst(5L, dist(pts)))
```

Index

∗ **reporting functions**
    print.Coreset, 21
    print.KCentre, 22
    print.MaxEntropy, 23
    print.MaxMeanSelection, 24
    print.MaxSum, 25
    summary.Coreset, 26

DropAdd, 3
DropAdd(), *7*, *9, 10*, *13*, *18–20*

ExactKCenter *(*ExactKCentre*)*, 5
ExactKCentre, 5
ExactKCentre(), *14, 15*, *22*
ExactMaxMin, 6
ExactMaxMin(), *6*, *10*, *13*, *20, 21*, *25*
ExactMaxSum, 8
ExactMaxSum(), *25*

FarFirst, 9
FarFirst(), *3*, *12–14*, *18–21*, *26*
format.KCentreExact *(*print.KCentre*)*, 22
format.KCentreSelection *(*print.KCentre*)*, 22
format.MaxEntropySelection *(*print.MaxEntropy*)*, 23
format.MaxMeanSelection *(*print.MaxMeanSelection*)*, 24
format.MaxMinSelection *(*print.Coreset*)*, 21
format.MaxSumSelection *(*print.MaxSum*)*, 25

Grasp, 11
Grasp(), *7*, *18–20*

KCenter *(*KCentre*)*, 13
KCenterRadius *(*KCentreRadius*)*, 14
KCentre, 13
KCentre(), *5, 6*, *14, 15*, *22*
KCentreRadius, 14
KCentreRadius(), *14*

MaxEntropy, 15
MaxEntropy(), *23*
MaxMean, 17
MaxMean(), *18, 19*, *24*
MeanDist, 18
MeanDist(), *18*
MinDist, 19
MinDist(), *15*, *19*

PickPoint, 20
PickPoint(), *9, 10*
print.Coreset, 21, *23–26*
print.KCentre, *22*, 22, *23–26*
print.KCentreExact *(*print.KCentre*)*, 22
print.KCentreSelection *(*print.KCentre*)*, 22
print.MaxEntropy, *22, 23*, 23, *24–26*
print.MaxEntropySelection *(*print.MaxEntropy*)*, 23
print.MaxMeanSelection, 24
print.MaxMeanSelection(), *18*, *22, 23*, *25, 26*
print.MaxMinSelection *(*print.Coreset*)*, 21
print.MaxMinSelection(), *4*, *7*, *12*, *26*
print.MaxSum, *22–24*, 25, *26*
print.MaxSumSelection *(*print.MaxSum*)*, 25

summary.Coreset, *22–25*, 26
summary.MaxMeanSelection *(*print.MaxMeanSelection*)*, 24
summary.MaxMinSelection *(*summary.Coreset*)*, 26