

Package: hySAINT (via r-universe)

October 9, 2026

Title Hybrid Genetic Algorithm and Simulated Annealing for Variable Selection with Interaction Effects

Version 2.3.0

Description A hybrid genetic algorithm combined with simulated annealing for variable selection in high-dimensional linear regression with two-way interaction effects. The package uses an indexed model formulation for efficient storage and minimizes the ABC (Adaptive Best-subset with Complexity) criterion from Ye and Yang (2019) [doi:10.1109/TIT.2019.2913417](https://doi.org/10.1109/TIT.2019.2913417). Features include: (1) support for all three heredity conditions (strong, weak, and no heredity), (2) adaptive temperature control in simulated annealing, (3) LASSO-based screening for ultra-high-dimensional problems, and (4) automatic sigma and lambda tuning. The package scales to problems with thousands of predictors.

License GPL-2

Encoding UTF-8

Depends R (>= 3.5.0)

Imports stats, Matrix, glmnet

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Language en-US

Author Leiyue Li [aut, cre], Zhengzhong You [aut], Zeya Wang [aut], Chenglong Ye [aut]

Maintainer Leiyue Li <lli289.git@gmail.com>

URL <https://github.com/lli289/hySAINT>

BugReports <https://github.com/lli289/hySAINT/issues>

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

NeedsCompilation no

Repository <https://lli289.r-universe.dev>

Date/Publication 2026-07-11 06:03:30 UTC
RemoteUrl <https://github.com/li289/hysaint>
RemoteRef HEAD
RemoteSha e89575feb8c202cf4afaebaceac4653d3e7be677

Contents

ABC	2
crossover	4
evaluate_individual	5
generate_initial_population	6
generate_inter_pool	7
hySAINT	8
mutate	11
screen_main_effects	12
Index	14

ABC	<i>ABC Evaluation Criterion</i>
-----	---------------------------------

Description

Computes the ABC (Adaptive Best-subset with Complexity) score for a fitted model following Ye and Yang (2019). Lower ABC scores indicate better model fit.

Usage

```
ABC(  
  X,  
  y,  
  heredity = "Strong",  
  sigma = NULL,  
  main_idx = NULL,  
  inter_idx = NULL,  
  lambda = 10,  
  p = NULL  
)
```

Arguments

X	Input data matrix of dimension n observations by p predictors.
y	Response variable. A numeric vector of length n.
heredity	Heredity constraint: "Strong", "Weak", or "No". Default is "Strong".
sigma	The standard deviation of noise. If NULL, it is estimated automatically.

main_idx	Indices of selected main effects (integers from 1 to p).
inter_idx	Interaction indices. Can be either: • A vector of linear indices (where interaction between X_i and X_j has index $p*(i-1) - i*(i-1)/2 + (j-i)$), or • A two-column matrix where each row contains the pair (i, j).
lambda	ABC penalty parameter. Must satisfy $\lambda \geq 5.1/\log(2) \approx 7.36$. Default is 10.
p	Total number of main effects. If NULL, uses ncol(X).

Details

The complexity term C_I depends on the heredity condition:

Strong heredity:

$$C_I = \log \binom{p}{k_1} + \log \binom{\binom{k_1}{2}}{k_2}$$

Weak heredity:

$$C_I = \log \binom{p}{k_1} + \log \binom{K}{k_2}$$

where $K = k_1 p - \binom{k_1}{2} - k_1$.

No heredity:

$$C_I = \log \binom{p}{k_1} + \log \binom{\binom{p}{2}}{k_2}$$

The function uses lchoose for numerical stability with large binomial coefficients.

Value

A list with components:

ABC	The ABC score of the fitted model.
SSE	Sum of squared errors.
r_I	Rank of the design matrix (number of parameters).
C_I	Complexity penalty term.
k1	Number of main effects in the model.
k2	Number of interactions in the model.

References

Ye, C. and Yang, Y. (2019). High-dimensional adaptive minimax sparse estimation with interactions. *IEEE Transactions on Information Theory*, 65(9), 5367-5379. doi:10.1109/TIT.2019.2913417

See Also

[hySAINT](#) for the main variable selection function.

Examples

```
set.seed(123)
n <- 100; p <- 10
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] + X[,2] + 0.5*X[,1]*X[,2] + rnorm(n, 0, 0.5)

# Compute ABC for a model with main effects 1, 2 and interaction 1:2
result <- ABC(X, y, heredity = "Strong", sigma = 0.5,
             main_idx = c(1, 2),
             inter_idx = matrix(c(1, 2), nrow = 1))
print(result$ABC)

# Compare with a smaller model (main effects only)
result2 <- ABC(X, y, heredity = "Strong", sigma = 0.5,
             main_idx = c(1, 2), inter_idx = NULL)
print(result2$ABC) # Should be higher (worse) since true model has interaction
```

crossover

Crossover Operator

Description

Combines two parent individuals to create one offspring individual for the genetic algorithm used by [hySAINT](#). The operator preserves the requested heredity constraint when carrying interaction terms into the child.

Usage

```
crossover(parent1, parent2, inter_pool, r1, r2, heredity, p)
```

Arguments

parent1	First parent individual, a list with main_idx and inter_idx.
parent2	Second parent individual, a list with main_idx and inter_idx.
inter_pool	Available interaction pool, usually generated by generate_inter_pool .
r1	Maximum number of main effects.
r2	Maximum number of interactions.
heredity	Heredity constraint: "Strong", "Weak", or "No".
p	Total number of main effects in the original data.

Value

An offspring individual represented as a list with integer vectors main_idx and inter_idx.

See Also

[generate_initial_population](#), [mutate](#), [evaluate_individual](#)

Examples

```
parent1 <- list(main_idx = c(1, 2, 3), inter_idx = c(1))
parent2 <- list(main_idx = c(2, 3, 4), inter_idx = c(2))
inter_pool <- generate_inter_pool(c(1, 2, 3, 4), p = 10, heredity = "Strong")

set.seed(1)
child <- crossover(parent1, parent2, inter_pool,
                  r1 = 3, r2 = 1, heredity = "Strong", p = 10)
child
```

evaluate_individual	<i>Evaluate an Individual by ABC Score</i>
---------------------	--

Description

Converts an individual from the genetic algorithm representation to model indices and evaluates it with [ABC](#).

Usage

```
evaluate_individual(individual, X, y, heredity, sigma, lambda, p)
```

Arguments

individual	Individual to evaluate, a list with main_idx and inter_idx.
X	Input data matrix.
y	Response vector.
heredity	Heredity constraint: "Strong", "Weak", or "No".
sigma	Noise standard deviation.
lambda	ABC penalty parameter.
p	Total number of main effects in the original data.

Value

Numeric ABC score for the candidate model.

See Also

[ABC](#), [generate_initial_population](#)

Examples

```
set.seed(1)
X <- matrix(rnorm(40), nrow = 10)
y <- X[, 1] + rnorm(10)
individual <- list(main_idx = c(1), inter_idx = integer(0))

evaluate_individual(individual, X, y,
                   heredity = "Strong", sigma = 1,
                   lambda = 5.1 / log(2), p = ncol(X))
```

generate_initial_population

Generate Initial Population for Genetic Algorithm

Description

Creates an initial population of candidate models for the genetic algorithm. Each individual represents a model with a random subset of main effects and interactions satisfying the heredity constraint. The first few individuals use top-ranked variables to seed the population with good solutions.

Usage

```
generate_initial_population(
  main_pool,
  inter_pool,
  r1,
  r2,
  pop_size,
  heredity,
  p
)
```

Arguments

main_pool	Ranked main effect candidates (from screen_main_effects).
inter_pool	Ranked interaction candidates (from generate_inter_pool).
r1	Maximum number of main effects per individual.
r2	Maximum number of interactions per individual.
pop_size	Number of individuals in the population.
heredity	Heredity constraint: "Strong", "Weak", or "No".
p	Total number of main effects in the original data.

Details

The function uses a biased sampling scheme where top-ranked variables (from LASSO screening) have higher probability of selection. This helps the algorithm converge faster by starting with promising candidates.

Value

A list of individuals, where each individual is a list with:

main_idx	Integer vector of selected main effect indices.
inter_idx	Integer vector of selected interaction linear indices.

See Also

[hySAINT](#) which calls this function, `crossover()` and `mutate()` for genetic operators.

Examples

```
# This function is typically called internally by hySAINT
# Example for illustration:
main_pool <- c(1, 2, 3, 5, 8, 10)
inter_pool <- matrix(c(1, 2, 101,
                      1, 3, 102,
                      2, 3, 103), ncol = 3, byrow = TRUE)
colnames(inter_pool) <- c("i", "j", "idx")

pop <- generate_initial_population(main_pool, inter_pool,
                                r1 = 4, r2 = 2,
                                pop_size = 10,
                                heredity = "Strong", p = 50)

length(pop) # 10 individuals
```

generate_inter_pool	<i>Generate Interaction Pool</i>
---------------------	----------------------------------

Description

Generates candidate interaction effects based on selected main effects and the heredity constraint. The pool size is controlled by the heredity condition:

- Strong: Only pairs where both variables are in main_idx
- Weak: Pairs where at least one variable is in main_idx
- No: All pairs among candidate main effects

Usage

```
generate_inter_pool(main_idx, p, heredity = "Strong", all_main_pool = NULL)
```

Arguments

main_idx	Indices of selected main effects.
p	Total number of main effects in the original data.
heredity	Heredity constraint: "Strong", "Weak", or "No". Default is "Strong".
all_main_pool	Full pool of candidate main effects. Used for weak heredity to determine which additional interactions to consider.

Value

A matrix with three columns:

i	First variable in the interaction pair.
j	Second variable in the interaction pair ($j > i$).
idx	Linear index of the interaction.

See Also

[screen_main_effects](#) for screening main effects, `rank_interactions()` for ranking interaction candidates.

Examples

```
# Strong heredity: only interactions among selected main effects
main_selected <- c(1, 2, 5, 8)
pool <- generate_inter_pool(main_selected, p = 100, heredity = "Strong")
print(nrow(pool)) # Should be choose(4, 2) = 6

# Weak heredity: more interactions allowed
pool_weak <- generate_inter_pool(main_selected, p = 100, heredity = "Weak",
                                all_main_pool = 1:20)
print(nrow(pool_weak)) # Larger than strong heredity
```

hySAINT

Hybrid Genetic Algorithm and Simulated Annealing for Variable Selection

Description

This is the main function of the **hySAINT** package. It implements a hybrid genetic algorithm combined with simulated annealing for variable selection in high-dimensional linear regression with two-way interaction effects. The algorithm minimizes the ABC (Adaptive Best-subset with Complexity) criterion from Ye and Yang (2019), which achieves selection consistency under all three heredity conditions.

Usage

```
hySAINT(
  X,
  y,
  heredity = c("Strong", "Weak", "No"),
  r1,
  r2,
  sigma = NULL,
  lambda = NULL,
  pop_size = 50,
```

```

    max_iter = 100,
    patience = 30,
    verbose = TRUE
)

```

Arguments

<code>X</code>	Input data matrix of dimension n observations by p predictors.
<code>y</code>	Response variable. A numeric vector of length n .
<code>heredity</code>	Heredity constraint: "Strong", "Weak", or "No". Default is "Strong". • Strong: An interaction $X_i * X_j$ can only be selected if both X_i and X_j are selected. • Weak: An interaction $X_i * X_j$ can only be selected if at least one of X_i or X_j is selected. • No: Interactions can be selected regardless of main effects.
<code>r1</code>	Maximum number of main effects to select.
<code>r2</code>	Maximum number of interaction effects to select.
<code>sigma</code>	The standard deviation of noise. If NULL (default), it is estimated automatically using the median absolute deviation (MAD) estimator.
<code>lambda</code>	ABC penalty parameter. If NULL (default), uses the theoretical minimum $\lambda = 5.1 / \log(2) \approx 7.36$ from Ye and Yang (2019).
<code>pop_size</code>	Population size for the genetic algorithm. Default is 50.
<code>max_iter</code>	Maximum number of GA iterations. Default is 100.
<code>patience</code>	Early stopping patience. The algorithm stops if no improvement is found for this many consecutive iterations. Default is 30.
<code>verbose</code>	Logical. Print progress messages? Default is TRUE.

Details

The algorithm proceeds in four main steps:

1. **Screening:** Main effects are screened using LASSO (via `glmnet`). This reduces the search space from p to a manageable number of candidates.
2. **Interaction Pool:** Candidate interactions are generated based on the heredity condition and ranked by partial correlation with the response.
3. **Genetic Algorithm:** A population of candidate models evolves through selection, crossover, and mutation operations.
4. **Simulated Annealing:** The mutation step uses a Metropolis acceptance criterion with adaptive temperature control, allowing the algorithm to escape local optima.

The ABC criterion balances goodness-of-fit with model complexity:

$$ABC(I) = SSE + 2r_I\sigma^2 + \lambda\sigma^2C_I$$

where r_I is the model rank and C_I is a heredity-specific complexity term derived from minimum description length theory.

Value

An object of class "hySAINT" with components:

main_effects	Indices of selected main effects (1 to p).
interactions	Matrix of selected interaction pairs (each row is a pair of indices).
ABC_score	Final ABC score of the selected model.
coefficients	Fitted OLS coefficients for the selected model.
sigma	Estimated or provided sigma value.
lambda	Lambda value used.
runtime	Total computation time in seconds.
convergence	List with convergence information including best scores per iteration.

References

Ye, C. and Yang, Y. (2019). High-dimensional adaptive minimax sparse estimation with interactions. *IEEE Transactions on Information Theory*, 65(9), 5367-5379. doi:10.1109/TIT.2019.2913417

See Also

[ABC](#) for computing the ABC criterion directly, [screen_main_effects](#) for the screening step, [generate_initial_population](#) for population initialization.

Examples

```
# Example 1: Low-dimensional case (p < n)
set.seed(123)
n <- 200; p <- 20
X <- matrix(rnorm(n * p), n, p)
# True model: y = X1 + X2 + X3 + X1*X2 + noise
y <- X[,1] + X[,2] + X[,3] + X[,1]*X[,2] + rnorm(n, 0, 0.5)

result <- hySAINT(X, y, heredity = "Strong", r1 = 5, r2 = 3,
                  max_iter = 50, verbose = FALSE)
print(result)

# Example 2: High-dimensional case (p > n)
## Not run:
set.seed(456)
n <- 100; p <- 500
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] + X[,2] + 0.5*X[,1]*X[,2] + rnorm(n, 0, 1)

result <- hySAINT(X, y, heredity = "Strong", r1 = 10, r2 = 5)
print(result)

## End(Not run)
```

mutate	<i>Mutation Operator</i>
--------	--------------------------

Description

Applies one stochastic mutation step to a candidate model in the genetic algorithm used by [hySAINT](#). Possible mutation moves include adding, removing, or swapping main effects and interactions while respecting the chosen heredity constraint.

Usage

```
mutate(individual, main_pool, inter_pool, r1, r2, heredity, p, mutation_rate = 0.3)
```

Arguments

individual	Individual to mutate, a list with main_idx and inter_idx.
main_pool	Available main-effect candidates.
inter_pool	Available interaction pool.
r1	Maximum number of main effects.
r2	Maximum number of interactions.
heredity	Heredity constraint: "Strong", "Weak", or "No".
p	Total number of main effects in the original data.
mutation_rate	Probability of applying a mutation. Default is 0.3.

Value

A mutated individual represented as a list with integer vectors main_idx and inter_idx.

See Also

[generate_initial_population](#), [crossover](#), [evaluate_individual](#)

Examples

```
individual <- list(main_idx = c(1, 2, 3), inter_idx = c(1))
main_pool <- 1:6
inter_pool <- generate_inter_pool(main_pool, p = 10, heredity = "Strong")

set.seed(1)
mutated <- mutate(individual, main_pool, inter_pool,
                  r1 = 4, r2 = 2, heredity = "Strong",
                  p = 10, mutation_rate = 1)

mutated
```

screen_main_effects *Screen Main Effects Using LASSO*

Description

Screens main effects using the LASSO regularization path. Variables that enter the model earlier and more frequently across the regularization path are ranked higher. This approach is more robust than simple correlation screening because it accounts for correlations among predictors.

Usage

```
screen_main_effects(X, y, n.lambda = 50, max.main = NULL)
```

Arguments

X	Input data matrix of dimension n by p.
y	Response variable. A numeric vector of length n.
n.lambda	Number of lambda values in the LASSO path. Default is 50.
max.main	Maximum number of main effects to return. If NULL (default), uses $\min(\text{floor}(n/\log(n)), p)$.

Details

The function fits a LASSO path using **glmnet** and ranks variables by:

1. Order of entry (earlier = more important)
2. Frequency of selection across the lambda path

If **glmnet** is not available, the function falls back to correlation-based screening.

Value

A list with components:

ranked_main	Indices of main effects ranked by importance (earlier entry in LASSO path and higher selection frequency).
selection_freq	Number of times each variable was selected across the lambda path.
first_entry	Index along the path where each variable first entered.
method	Screening method used ("glmnet" or "correlation" as fallback).

See Also

[hySAINT](#) which uses this function internally, [generate_inter_pool](#) for generating interaction candidates.

Examples

```
set.seed(123)
n <- 100; p <- 50
X <- matrix(rnorm(n * p), n, p)
y <- X[,1] + X[,2] + X[,3] + rnorm(n)

screened <- screen_main_effects(X, y, max.main = 20)
print(screened$ranked_main[1:10]) # Top 10 main effects
```

Index

ABC, [2](#), [5](#), [10](#)

crossover, [4](#), [11](#)

evaluate_individual, [4](#), [5](#), [11](#)

generate_initial_population, [4](#), [5](#), [6](#), [10](#),
[11](#)

generate_inter_pool, [4](#), [7](#), [12](#)

hySAINT, [3](#), [4](#), [7](#), [8](#), [11](#), [12](#)

mutate, [4](#), [11](#)

screen_main_effects, [8](#), [10](#), [12](#)