Diagnostics Supplemental Material

Jose Guadalupe Hernandez

2023-01-30

Contents

Chapter 1

Introduction

This is the supplemental material associated with our 2022 ECJ contribution entitled, *A suite of diagnostic metrics for characterizing selection schemes*. Preprint here.

1.1 About our supplemental material

This supplemental material is hosted on GitHub using GitHub pages. The source code and configuration files used to generate this supplemental material can be found in this GitHub repository. We compiled our data analyses and supplemental documentation into this nifty web-accessible book using bookdown.

Our supplemental material includes the following paper figures and statistics:

- Exploitation rate results (Section 2)
- Ordered exploitation results (Section 3)
- Contradictory objectives results (Section 4)
- Multi-path exploration results (Section **??**)
- Multi-valley crossing results (Section 2.5)

Additionally, our supplemental material includes the results from parameter tuning selection schemes:

- Truncation selection (Section **??**)
- Tournament selection sharing (Section **??**)
- Genotypic fitness sharing (Section **??**)
- Phenotypic fitness sharing (Section **??**)
- Nondominated sorting (Section **??**)
- Novelty search (Section **??**)

1.2 Contributing authors

- Jose Guadalupe Hernandez
- Alexander Lalejini
- Charles Ofria

1.3 Research overview

Abstract:

Evolutionary algorithms typically consist of multiple interacting components, where each component influences an algorithm's problem-solving abilities. Understanding how each component of an evolutionary algorithm influences problem-solving success can improve our ability to target particular problem domains. Benchmark suites provide insights into an evolutionary algorithm's problem-solving capabilities, but benchmarking problems often have complex search space topologies, making it difficult to isolate and test an algorithm's strengths and weaknesses. Our work focuses on diagnosing selection schemes, which identify individuals to contribute genetic material to the next generation, thus driving an evolutionary algorithm's search strategy. We introduce four diagnostics for empirically testing the strengths and weaknesses of selection schemes: the exploitation rate diagnostic, ordered exploitation rate diagnostic, contradictory objectives diagnostic, and the multi-path exploration diagnostic. Each diagnostic is a handcrafted search space designed to isolate and measure the relative exploitation and exploration characteristics of selection schemes. Here, we use our diagnostics to evaluate six population selection methods: truncation selection, tournament selection, fitness sharing, lexicase selection, nondominated sorting, and novelty search. Expectedly, tournament and truncation selection excelled at gradient exploitation but poorly explored search spaces, while novelty search excelled at exploration but failed to exploit gradients. Fitness sharing performed poorly across all diagnostics, suggesting poor overall exploitation and exploration abilities. Nondominated sorting was best for maintaining diverse populations comprised of individuals inhabiting multiple optima, but struggled to effectively exploit gradients. Lexicase selection balanced search space exploration without sacrificing exploitation, generally performing well across diagnostics. Our work demonstrates the value of diagnostics for building a deeper understanding of selection schemes, which can then be used to improve or develop new selection methods.

1.4 Computer Setup

These analyses were conducted in the following computing environment:

```
print(version)
```

```
##                _
## platform       x86_64-pc-linux-gnu
## arch           x86_64
## os             linux-gnu
## system         x86_64, linux-gnu
## status         Patched
## major          4
## minor          2.2
## year           2022
## month          11
## day            10
## svn rev        83330
## language       R
## version.string R version 4.2.2 Patched (2022-11-10 r83330)
## nickname       Innocent and Trusting
```

1.5 Experimental setup

Setting up required variables variables.

```
# includes

library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2
## --

## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.1.8     v stringr 1.5.0
## v tidyr   1.3.0     v forcats 1.0.0
## v readr   2.1.3
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::arrange()   masks plyr::arrange()
## x purrr::compact()   masks plyr::compact()
## x dplyr::count()     masks plyr::count()
## x dplyr::desc()      masks plyr::desc()
## x dplyr::failwith()  masks plyr::failwith()
## x dplyr::filter()    masks stats::filter()
## x dplyr::id()        masks plyr::id()
## x dplyr::lag()       masks stats::lag()
## x dplyr::mutate()    masks plyr::mutate()
## x dplyr::rename()    masks plyr::rename()
## x dplyr::summarise() masks plyr::summarise()
## x dplyr::summarize() masks plyr::summarize()
```

```
# graph variables
SHAPE = c(5,3,1,2,6,0,4,20,1)
cb_palette <- c('#332288','#88CCEE','#EE7733','#EE3377','#117733','#882255','#44AA99',
mvc_col = c('#1A85FF','#D41159')
TSIZE = 26
p_theme <- theme(
  text = element_text(size = 28),
  plot.title = element_text( face = "bold", size = 22, hjust=0.5),
  panel.border = element_blank(),
  panel.grid.minor = element_blank(),
  legend.title=element_text(size=22),
  legend.text=element_text(size=23),
  axis.title = element_text(size=23),
  axis.text = element_text(size=22),
  legend.position="bottom",
  panel.background = element_rect(fill = "#f1f2f5",
                                  colour = "white",
                                  size = 0.5, linetype = "solid")
)
```

```
## Warning: The `size` argument of `element_rect()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
```

```
# default variables
REPLICATES = 50
DIMENSIONALITY = 100

# selection scheme related stuff
ACRON = tolower(c('TRU','TOR','LEX','GFS','PFS','NDS','NOV','RAN'))
NAMES = c('Truncation (tru)','Tournament (tor)','Lexicase (lex)', 'Genotypic Fitness S
SCHEME = c('TRUNCATION','TOURNAMENT','LEXICASE','FITSHARING_G','FITSHARING_P','NONDOMI
ORDER = c('Truncation (tru)','Tournament (tor)','Lexicase (lex)', 'Genotypic Fitness S
```

```r
# selection scheme parameters
TR_LIST = c(1, 2, 4, 8, 16, 32, 64, 128, 256)
TS_LIST = c(2, 4, 8, 16, 32, 64, 128, 256)
FS_LIST = c(0.0, 0.1, 0.3, 0.6, 1.2, 2.5, 5.0)
ND_LIST = c(0.0, 0.1, 0.3, 0.6, 1.2, 2.5, 5.0)
NS_LIST = c(1, 2, 4, 8, 15, 30)

# selection scheme parameter we are looking for
PARAM = c('8', '8', '0.0', '0.3', '0.3', '0.3', '15', '1')

# for diagnostic loops
DIAGNOSTIC = tolower(c('EXPLOITATION_RATE', 'ORDERED_EXPLOITATION', 'CONTRADICTORY_OBJECTIVES', '

# data diractory for gh-pages
DATA_DIR = '/opt/ECJ-2022-suite-of-diagnostics-for-selection-schemes/DATA-FINAL/'

######################################################################################
# go through each diagnostic and collect over time data for cross comparison (cc)
print('Collecting over time data...')
```

```
## [1] "Collecting over time data..."
```

```r
cc_over_time = data.frame()
cc_over_time_mvc = data.frame()
for(diagnostic in DIAGNOSTIC)
{
  print(paste('DIAGNOSTIC',diagnostic))
  for(i in 1:8)
  {
    print(paste('SCHEME:',SCHEME[i]))
    dir = paste(DATA_DIR,'NO-MVC/',SCHEME[i],'/over-time-',diagnostic,'-', tolower(SCHEME[i]), '.
    dir_mvc = paste(DATA_DIR,'MVC/',SCHEME[i],'/over-time-',diagnostic,'-', tolower(SCHEME[i]), '

    # read csv
    df = read.csv(dir, header = TRUE, stringsAsFactors = FALSE)
    df_mvc = read.csv(dir_mvc, header = TRUE, stringsAsFactors = FALSE)

    # add names/tags
    df$acron = ACRON[i]
    df$`Selection\nScheme` = NAMES[i]
    df$diagnostic = diagnostic

    df_mvc$acron = ACRON[i]
    df_mvc$`Selection\nScheme` = NAMES[i]
    df_mvc$diagnostic = diagnostic
    # add to cc_over_time data frame
    if(i == 3)
    {
      cc_over_time = rbind(cc_over_time, df)
      cc_over_time_mvc = rbind(cc_over_time_mvc, df_mvc)
    }
    else
    {
      cc_over_time = rbind(cc_over_time, filter(df, trt == PARAM[i]))
      cc_over_time_mvc = rbind(cc_over_time_mvc, filter(df_mvc, trt == PARAM[i]))
    }
  }
  rm(df);  rm(df_mvc);  rm(dir);   rm(dir_mvc)
}
```

```
## [1] "DIAGNOSTIC exploitation_rate"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
## [1] "DIAGNOSTIC ordered_exploitation"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
## [1] "DIAGNOSTIC contradictory_objectives"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
## [1] "DIAGNOSTIC multipath_exploration"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
```

```
cc_over_time$`Selection\nScheme` <- factor(cc_over_time$`Selection\nScheme`, levels = ORDER)
cc_over_time$acron <- factor(cc_over_time$acron, levels = ACRON)
cc_over_time$uni_str_pos = cc_over_time$uni_str_pos + cc_over_time$arc_acti_gene - cc_over_time$c
cc_over_time = subset(cc_over_time, select = -c(trt,pop_fit_avg,archive_cnt,pmin,pareto_cnt,arc_a

cc_over_time_mvc$`Selection\nScheme` <- factor(cc_over_time$`Selection\nScheme`, levels = ORDER)
cc_over_time_mvc$acron <- factor(cc_over_time$acron, levels = ACRON)
cc_over_time_mvc$uni_str_pos = cc_over_time_mvc$uni_str_pos + cc_over_time_mvc$arc_acti_gene - cc
cc_over_time_mvc = subset(cc_over_time_mvc, select = -c(trt,pop_fit_avg,archive_cnt,pmin,pareto_c


######################################################################################
# go through each diagnostic and collect best over time for cross comparison (cc)
cc_best = data.frame()
cc_best_mvc = data.frame()
for(diagnostic in DIAGNOSTIC)
{
  print(paste('DIAGNOSTIC',diagnostic))
  for(i in 1:8)
  {
    print(paste('SCHEME:',SCHEME[i]))
    dir = paste(DATA_DIR,'NO-MVC/',SCHEME[i],'/best-',diagnostic,'-', tolower(SCHEME[i]), '.csv',
    dir_mvc = paste(DATA_DIR,'MVC/',SCHEME[i],'/best-',diagnostic,'-', tolower(SCHEME[i]), '.csv'

    # read csv
    df = read.csv(dir, header = TRUE, stringsAsFactors = FALSE)
    df_mvc = read.csv(dir_mvc, header = TRUE, stringsAsFactors = FALSE)

    # add names/tags
    df$acron = ACRON[i]
    df$`Selection\nScheme` = NAMES[i]
    df$diagnostic = diagnostic
    df = subset(df, select = -c(Diagnostic,SEL) )

    df_mvc$acron = ACRON[i]
    df_mvc$`Selection\nScheme` = NAMES[i]
    df_mvc$diagnostic = diagnostic
    df_mvc = subset(df_mvc, select = -c(Diagnostic,SEL) )
    # add to cc_over_time data frame
    if(i == 3)
    {
      cc_best = rbind(cc_best, df)
      cc_best_mvc = rbind(cc_best_mvc, df_mvc)
    }
    else
    {
      cc_best = rbind(cc_best, filter(df, trt == PARAM[i]))
      cc_best_mvc = rbind(cc_best_mvc, filter(df_mvc, trt == PARAM[i]))
    }
  }
  rm(df);  rm(df_mvc);  rm(dir);   rm(dir_mvc)
}
```

```
## [1] "DIAGNOSTIC exploitation_rate"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
## [1] "DIAGNOSTIC ordered_exploitation"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
## [1] "DIAGNOSTIC contradictory_objectives"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
## [1] "DIAGNOSTIC multipath_exploration"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
```

```
cc_best$acron <- factor(cc_best$acron, levels = ACRON)
cc_best = subset(cc_best, select = -c(trt,gen))
cc_best = filter(cc_best, col == 'pop_fit_max' | col == 'pop_uni_obj')

cc_best_mvc$acron <- factor(cc_best_mvc$acron, levels = ACRON)
cc_best_mvc = subset(cc_best_mvc, select = -c(trt,gen))
cc_best_mvc = subset(cc_best_mvc, col == 'pop_fit_max' | col == 'pop_uni_obj')


########################################################################################
# get generation a satisfactory solution is found for cross comparison (cc)
cc_ssf  = data.frame()
for(diagnostic in DIAGNOSTIC)
{
  if(diagnostic == 'contradictory_objectives' | diagnostic == 'multipath_exploration')
  {next}

  print(paste('DIAGNOSTIC',diagnostic))
  for(i in 1:8)
  {
    print(paste('SCHEME:',SCHEME[i]))
    dir = paste(DATA_DIR,'NO-MVC/',SCHEME[i],'/ssf-',diagnostic,'-', tolower(SCHEME[i]), '.csv',

    # read csv
    df = read.csv(dir, header = TRUE, stringsAsFactors = FALSE)

    # add names/tags
    df$acron = ACRON[i]
    df$`Selection\nScheme` = NAMES[i]
    df$diagnostic = diagnostic
    df = subset(df, select = -c(Diagnostic,SEL) )

    # add to cc_over_time data frame
    if(i == 3)
    {
      cc_ssf = rbind(cc_ssf, df)
    }
    else
    {
      cc_ssf = rbind(cc_ssf, filter(df, trt == PARAM[i]))
    }
  }
  rm(df); rm(dir);
}
```

```
## [1] "DIAGNOSTIC exploitation_rate"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
## [1] "DIAGNOSTIC ordered_exploitation"
## [1] "SCHEME: TRUNCATION"
## [1] "SCHEME: TOURNAMENT"
## [1] "SCHEME: LEXICASE"
## [1] "SCHEME: FITSHARING_G"
## [1] "SCHEME: FITSHARING_P"
## [1] "SCHEME: NONDOMINATEDSORTING"
## [1] "SCHEME: NOVELTY"
## [1] "SCHEME: TOURNAMENT"
```

```
cc_ssf$acron <- factor(cc_ssf$acron, levels = ACRON)
cc_ssf = subset(cc_ssf, select = -c(trt))


######################################################################################
# go through each scheme and collect over time data
ss_over_time = data.frame()
ss_over_time_mvc = data.frame()
for(i in 1:8)
{
  # add to cc_over_time data frame
  if(i == 3 | i == 8)
  {
    next
  }
  print(SCHEME[i])
  for(diagnostic in DIAGNOSTIC)
  {
    dir = paste(DATA_DIR,'NO-MVC/',SCHEME[i],'/over-time-',diagnostic,'-', tolower(SCH
    dir_mvc = paste(DATA_DIR,'MVC/',SCHEME[i],'/over-time-',diagnostic,'-', tolower(SC
    # read csv
    df = read.csv(dir, header = TRUE, stringsAsFactors = FALSE)
    df_mvc = read.csv(dir_mvc, header = TRUE, stringsAsFactors = FALSE)

    # add names/tags
    df$acron = ACRON[i]
    df$diagnostic = diagnostic

    df_mvc$acron = ACRON[i]
    df_mvc$diagnostic = diagnostic

    ss_over_time = rbind(ss_over_time, df)
    ss_over_time_mvc = rbind(ss_over_time_mvc,df_mvc)
  }
  rm(df);  rm(df_mvc);  rm(dir);   rm(dir_mvc)
}
```

```
## [1] "TRUNCATION"
## [1] "TOURNAMENT"
## [1] "FITSHARING_G"
## [1] "FITSHARING_P"
## [1] "NONDOMINATEDSORTING"
## [1] "NOVELTY"
```

```
# remove unused data

ss_over_time$uni_str_pos = ss_over_time$uni_str_pos + ss_over_time$arc_acti_gene - ss_over_time$c
ss_over_time = subset(ss_over_time, select = -c(pop_fit_avg,archive_cnt,pmin,pareto_cnt,arc_acti_

ss_over_time_mvc$uni_str_pos = ss_over_time_mvc$uni_str_pos + ss_over_time_mvc$arc_acti_gene - ss
ss_over_time_mvc = subset(ss_over_time_mvc, select = -c(pop_fit_avg,archive_cnt,pmin,pareto_cnt,a

## tournament data frames
tor_ot <- data.frame()
tor_ot <- filter(ss_over_time, acron == 'tor' & trt != 1)
tor_ot$T <- factor(tor_ot$trt, levels = TS_LIST)
tor_ot <- subset(tor_ot, select = -c(acron,trt))

## truncation data frames
tru_ot <- data.frame()
tru_ot <- filter(ss_over_time, acron == 'tru')
tru_ot$T <- factor(tru_ot$trt, levels = TR_LIST)
tru_ot <- subset(tru_ot, select = -c(acron,trt))

## genotypic fitness sharing data frames
gfs_ot <- data.frame()
gfs_ot <- filter(ss_over_time, acron == 'gfs')
gfs_ot$Sigma <- factor(gfs_ot$trt, levels = FS_LIST)
gfs_ot <- subset(gfs_ot, select = -c(acron,trt))

## phenotypic fitness sharing data frames
pfs_ot <- data.frame()
pfs_ot <- filter(ss_over_time, acron == 'pfs')
pfs_ot$Sigma <- factor(pfs_ot$trt, levels = FS_LIST)
pfs_ot <- subset(pfs_ot, select = -c(acron,trt))


## nodominated sorting data frames
nds_ot <- data.frame()
nds_ot <- filter(ss_over_time, acron == 'nds')
nds_ot$Sigma <- factor(nds_ot$trt, levels = ND_LIST)
nds_ot <- subset(nds_ot, select = -c(acron,trt))

## novelty search data frames
nov_ot <- data.frame()
nov_ot <- filter(ss_over_time, acron == 'nov' & trt != 0)
nov_ot$K <- factor(nov_ot$trt, levels = NS_LIST)
nov_ot <- subset(nov_ot, select = -c(acron,trt))

## tournament data frames mvc
tor_ot_mvc <- data.frame()
tor_ot_mvc <- filter(ss_over_time_mvc, acron == 'tor' & trt != 1)
tor_ot_mvc$T <- factor(tor_ot_mvc$trt, levels = TS_LIST)
tor_ot_mvc <- subset(tor_ot_mvc, select = -c(acron,trt))

## truncation data frames mvc
tru_ot_mvc <- data.frame()
tru_ot_mvc <- filter(ss_over_time_mvc, acron == 'tru')
tru_ot_mvc$T <- factor(tru_ot_mvc$trt, levels = TR_LIST)
tru_ot_mvc <- subset(tru_ot_mvc, select = -c(acron,trt))

## genotypic fitness sharing data frames mvc
gfs_ot_mvc <- data.frame()
gfs_ot_mvc <- filter(ss_over_time_mvc, acron == 'gfs')
gfs_ot_mvc$Sigma <- factor(gfs_ot_mvc$trt, levels = FS_LIST)
gfs_ot_mvc <- subset(gfs_ot_mvc, select = -c(acron,trt))

## phenotypic fitness sharing data frames mvc
pfs_ot_mvc <- data.frame()
pfs_ot_mvc <- filter(ss_over_time_mvc, acron == 'pfs')
pfs_ot_mvc$Sigma <- factor(pfs_ot_mvc$trt, levels = FS_LIST)
```

```
pfs_ot_mvc <- subset(pfs_ot_mvc, select = -c(acron,trt))


## nodominated sorting data frames mvc
nds_ot_mvc <- data.frame()
nds_ot_mvc <- filter(ss_over_time_mvc, acron == 'nds')
nds_ot_mvc$Sigma <- factor(nds_ot_mvc$trt, levels = ND_LIST)
nds_ot_mvc <- subset(nds_ot_mvc, select = -c(acron,trt))

## novelty search data frames mvc
nov_ot_mvc <- data.frame()
nov_ot_mvc <- filter(ss_over_time_mvc, acron == 'nov' & trt != 0)
nov_ot_mvc$K <- factor(nov_ot_mvc$trt, levels = NS_LIST)
nov_ot_mvc <- subset(nov_ot_mvc, select = -c(acron,trt))

# clean up
rm(ss_over_time_mvc)
rm(ss_over_time)


#######################################################################################
# go through each scheme and collect best data
ss_best = data.frame()
ss_best_mvc = data.frame()
for(i in 1:8)
{
  # add to cc_best data frame
  if(i == 3 | i == 8)
  {
    next
  }
  print(SCHEME[i])
  for(diagnostic in DIAGNOSTIC)
  {
    dir = paste(DATA_DIR,'NO-MVC/',SCHEME[i],'/best-',diagnostic,'-', tolower(SCHEME[i]), '.csv',
    dir_mvc = paste(DATA_DIR,'MVC/',SCHEME[i],'/best-',diagnostic,'-', tolower(SCHEME[i]), '.csv'

    # read csv
    df = read.csv(dir, header = TRUE, stringsAsFactors = FALSE)
    df_mvc = read.csv(dir_mvc, header = TRUE, stringsAsFactors = FALSE)

    # add names/tags
    df$acron = ACRON[i]
    df$diagnostic = diagnostic
    df_mvc$acron = ACRON[i]
    df_mvc$diagnostic = diagnostic

    ss_best = rbind(ss_best, df)
    ss_best_mvc = rbind(ss_best_mvc,df_mvc)
  }
  rm(df);  rm(df_mvc);  rm(dir);   rm(dir_mvc)
}
```

```
## [1] "TRUNCATION"
## [1] "TOURNAMENT"
## [1] "FITSHARING_G"
## [1] "FITSHARING_P"
## [1] "NONDOMINATEDSORTING"
## [1] "NOVELTY"
```

```
# removed unused data
ss_best = subset(ss_best, select = -c(gen))
ss_best = filter(ss_best, col == 'pop_fit_max' | col == 'pop_uni_obj')

ss_best_mvc = subset(ss_best_mvc, select = -c(gen))
ss_best_mvc = filter(ss_best_mvc, col == 'pop_fit_max' | col == 'pop_uni_obj')

## tournament data frames
tor_best <- data.frame()
tor_best <- filter(ss_best, acron == 'tor' & trt != 1)
tor_best$T <- factor(tor_best$trt, levels = TS_LIST)
tor_best <- subset(tor_best, select = -c(acron,trt))

## truncation data frames
tru_best <- data.frame()
tru_best <- filter(ss_best, acron == 'tru')
tru_best$T <- factor(tru_best$trt, levels = TR_LIST)
tru_best <- subset(tru_best, select = -c(acron,trt))

## genotypic fitness sharing data frames
gfs_best <- data.frame()
gfs_best <- filter(ss_best, acron == 'gfs')
gfs_best$Sigma <- factor(gfs_best$trt, levels = FS_LIST)
gfs_best <- subset(gfs_best, select = -c(acron,trt))

## phenotypic fitness sharing data frames
pfs_best <- data.frame()
pfs_best <- filter(ss_best, acron == 'pfs')
pfs_best$Sigma <- factor(pfs_best$trt, levels = FS_LIST)
pfs_best <- subset(pfs_best, select = -c(acron,trt))
```

```
## nodominated sorting data frames
nds_best <- data.frame()
nds_best <- filter(ss_best, acron == 'nds')
nds_best$Sigma <- factor(nds_best$trt, levels = ND_LIST)
nds_best <- subset(nds_best, select = -c(acron,trt))

## novelty search data frames
nov_best <- data.frame()
nov_best <- filter(ss_best, acron == 'nov' & trt != 0)
nov_best$K <- factor(nov_best$trt, levels = NS_LIST)
nov_best <- subset(nov_best, select = -c(acron,trt))

## tournament data frames mvc
tor_best_mvc <- data.frame()
tor_best_mvc <- filter(ss_best_mvc, acron == 'tor' & trt != 1)
tor_best_mvc$T <- factor(tor_best_mvc$trt, levels = TS_LIST)
tor_best_mvc <- subset(tor_best_mvc, select = -c(acron,trt))

## truncation data frames mvc
tru_best_mvc <- data.frame()
tru_best_mvc <- filter(ss_best_mvc, acron == 'tru')
tru_best_mvc$T <- factor(tru_best_mvc$trt, levels = TR_LIST)
tru_best_mvc <- subset(tru_best_mvc, select = -c(acron,trt))

## genotypic fitness sharing data frames mvc
gfs_best_mvc <- data.frame()
gfs_best_mvc <- filter(ss_best_mvc, acron == 'gfs')
gfs_best_mvc$Sigma <- factor(gfs_best_mvc$trt, levels = FS_LIST)
gfs_best_mvc <- subset(gfs_best_mvc, select = -c(acron,trt))

## phenotypic fitness sharing data frames mvc
pfs_best_mvc <- data.frame()
pfs_best_mvc <- filter(ss_best_mvc, acron == 'pfs')
pfs_best_mvc$Sigma <- factor(pfs_best_mvc$trt, levels = FS_LIST)
pfs_best_mvc <- subset(pfs_best_mvc, select = -c(acron,trt))


## nodominated sorting data frames mvc
nds_best_mvc <- data.frame()
nds_best_mvc <- filter(ss_best_mvc, acron == 'nds')
nds_best_mvc$Sigma <- factor(nds_best_mvc$trt, levels = ND_LIST)
nds_best_mvc <- subset(nds_best_mvc, select = -c(acron,trt))

## novelty search data frames mvc
```

```
nov_best_mvc <- data.frame()
nov_best_mvc <- filter(ss_best_mvc, acron == 'nov' & trt != 0)
nov_best_mvc$K <- factor(nov_best_mvc$trt, levels = NS_LIST)
nov_best_mvc <- subset(nov_best_mvc, select = -c(acron,trt))

# clean up
rm(ss_best_mvc)
rm(ss_best)


######################################################################################
# go through each scheme and collect satisfactory solution found

#Tournament
exp_dir = paste(DATA_DIR,'NO-MVC/TOURNAMENT/ssf-exploitation_rate-tournament.csv', sep
ord_dir = paste(DATA_DIR,'NO-MVC/TOURNAMENT/ssf-ordered_exploitation-tournament.csv', s
# read csv
exp_df = read.csv(exp_dir, header = TRUE, stringsAsFactors = FALSE)
ord_df = read.csv(ord_dir, header = TRUE, stringsAsFactors = FALSE)
# remove data
exp_df = subset(exp_df, select = -c(SEL))
exp_df = filter(exp_df, trt != 1)
ord_df = subset(ord_df, select = -c(SEL))
ord_df = filter(ord_df, trt != 1)
# combine
tru_ssf = rbind(exp_df,ord_df)

#Truncation
exp_dir = paste(DATA_DIR,'NO-MVC/TRUNCATION/ssf-exploitation_rate-truncation.csv', sep
ord_dir = paste(DATA_DIR,'NO-MVC/TRUNCATION/ssf-ordered_exploitation-truncation.csv', s
# read csv
exp_df = read.csv(exp_dir, header = TRUE, stringsAsFactors = FALSE)
ord_df = read.csv(ord_dir, header = TRUE, stringsAsFactors = FALSE)
# remove data
exp_df = subset(exp_df, select = -c(SEL))
exp_df = filter(exp_df, trt != 1)
ord_df = subset(ord_df, select = -c(SEL))
ord_df = filter(ord_df, trt != 1)
# combine
tru_ssf = rbind(exp_df,ord_df)

#final clean up
rm(i,exp_dir,ord_dir,exp_df,ord_df)
```

Chapter 2

Exploitation rate results

Here we present the results for **best performances** found by each selection scheme replicate on the exploitation rate diagnostic. Best performance found refers to the largest average trait score found in a given population. Note that performance values fall between 0.0 and 100.0.

2.1 Analysis dependencies

```r
library(ggplot2)
library(cowplot)
library(dplyr)
library(PupillometryR)
library(sdamr)
```

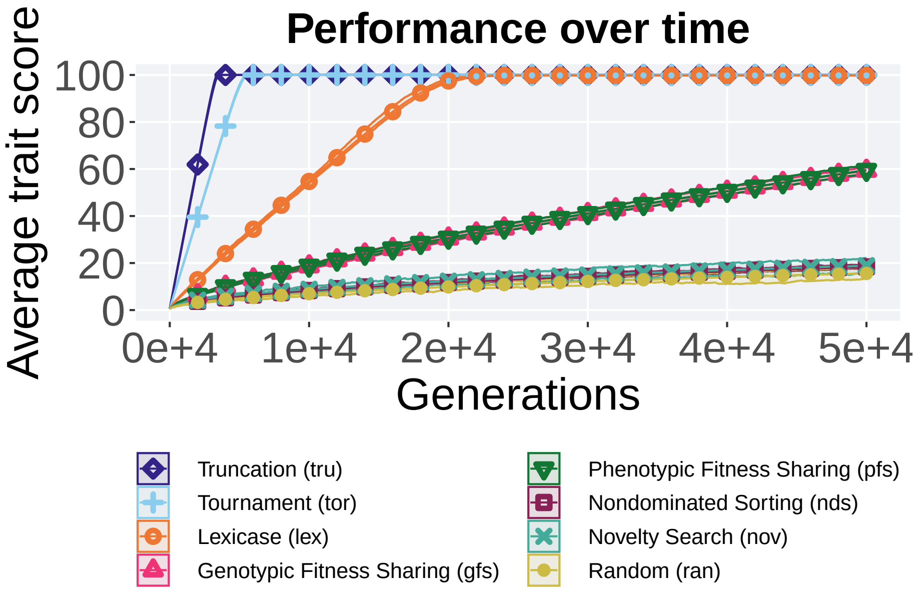
2.2 Performance over time

Best performance in a population over time.

```r
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'exploitation_rate') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

```r
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")

  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Performance over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```

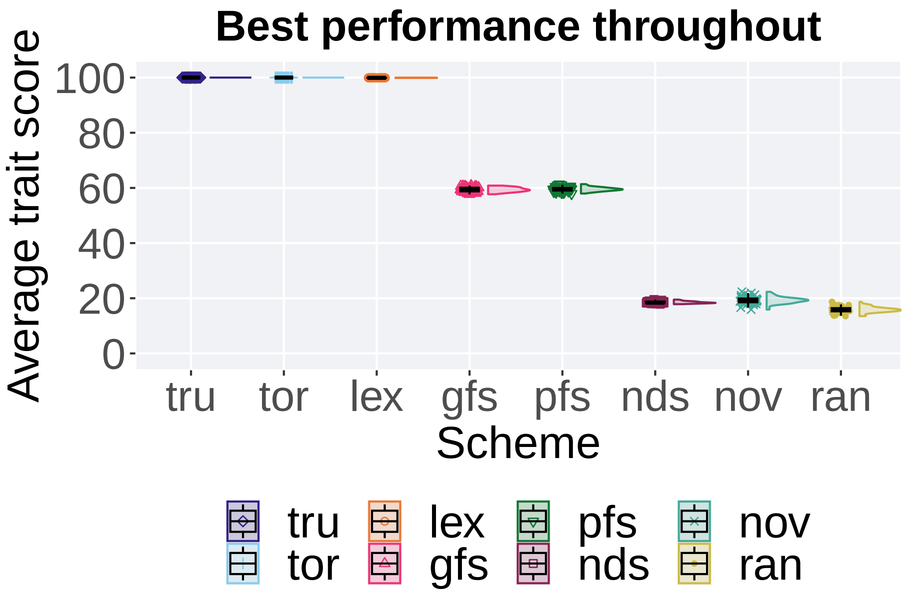


2.3 Best performance throughout

Best performance found throughout 50,000 generations.

```
### best performance throughout
filter(cc_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape = acron))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Average trait score",
    limits=c(-1, 101),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best performance throughout')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```

```
## Warning: Using the `size` aesthietic with geom_polygon was deprecated in ggplot2 3.4
## i Please use the `linewidth` aesthetic instead.
```



2.3.1 Stats

Summary statistics for the best performance.

```
#get data & summarize
performance = filter(cc_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate')
performance$acron = factor(performance$acron, levels = c('tru', 'tor', 'lex', 'gfs', 'p
performance %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val / DIMENSIONALITY, na.rm = TRUE),
    median = median(val / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(val / DIMENSIONALITY, na.rm = TRUE),
    max = max(val / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(val / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max    IQR
##   <fct> <int>  <int> <dbl>  <dbl> <dbl> <dbl>  <dbl>
## 1 tru      50      0 100    100   100   100   0
## 2 tor      50      0 100    100   100   100   0
## 3 lex      50      0  99.9   99.9  99.9  99.9 0.0137
## 4 gfs      50      0  57.7   59.3  59.4  60.8 1.31
## 5 pfs      50      0  58.0   59.5  59.5  61.4 0.908
## 6 nov      50      0  15.9   19.2  19.3  22.3 1.34
## 7 nds      50      0  17.9   18.4  18.5  19.5 0.516
## 8 ran      50      0  13.5   15.9  15.9  18.7 1.15
```

Kruskal–Wallis test provides evidence of statistical differences.

```
kruskal.test(val ~ acron, data = performance)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 384.91, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction.

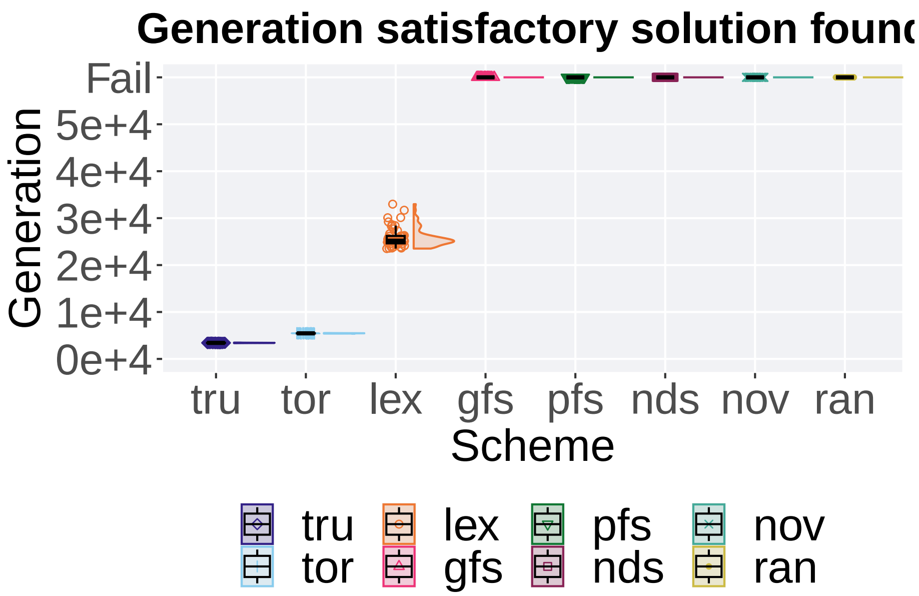
```
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bonferroni",
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  performance$val and performance$acron
##
##     tru     tor     lex     gfs     pfs     nov     nds
## tor 1e+00   -       -       -       -       -       -
## lex < 2e-16 < 2e-16 -       -       -       -       -
## gfs < 2e-16 < 2e-16 < 2e-16 -       -       -       -
## pfs < 2e-16 < 2e-16 < 2e-16 1e+00   -       -       -
## nov < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -       -
## nds < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 6e-04   -
## ran < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 1.9e-15 7.9e-16
##
## P value adjustment method: bonferroni
```

2.4 Generation satisfactory solution found

First generation a satisfactory solution is found throughout the 50,000 generations.

```
filter(cc_ssf, diagnostic == 'exploitation_rate') %>%
  ggplot(., aes(x = acron, y = Generations , color = acron, fill = acron, shape = acron
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Generation",
    limits=c(0, 60001),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000, 60000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4", "Fail")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Generation satisfactory solution found')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



2.4.1 Stats

Summary statistics for the first generation a satisfactory solution is found.

```
ssf = filter(cc_ssf, diagnostic == 'exploitation_rate'  & Generations < 60000)
ssf$acron = factor(ssf$acron, levels = c('tru', 'tor', 'lex'))
ssf %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(Generations)),
    min = min(Generations, na.rm = TRUE),
    median = median(Generations, na.rm = TRUE),
    mean = mean(Generations, na.rm = TRUE),
    max = max(Generations, na.rm = TRUE),
    IQR = IQR(Generations, na.rm = TRUE)
  )
```

```
## # A tibble: 3 x 8
##   acron count na_cnt   min median   mean   max    IQR
##   <fct> <int>  <int> <int>  <dbl>  <dbl> <int>  <dbl>
## 1 tru      50      0  3357   3420  3421.  3481   34.2
## 2 tor      50      0  5403   5457  5453.  5519   51.8
## 3 lex      50      0 23514  25190 25857. 32980 1581
```

Kruskal–Wallis test provides evidence of difference amoung selection schemes.

```
kruskal.test(Generations ~ acron, data = ssf)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Generations by acron
## Kruskal-Wallis chi-squared = 132.46, df = 2, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction.

```
pairwise.wilcox.test(x = ssf$Generations, g = ssf$acron, p.adjust.method = "bonferroni"
                     paired = FALSE, conf.int = FALSE, alternative = 'g')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  ssf$Generations and ssf$acron
##
##     tru    tor
## tor <2e-16 -
## lex <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

2.5 Multi-valley crossing results

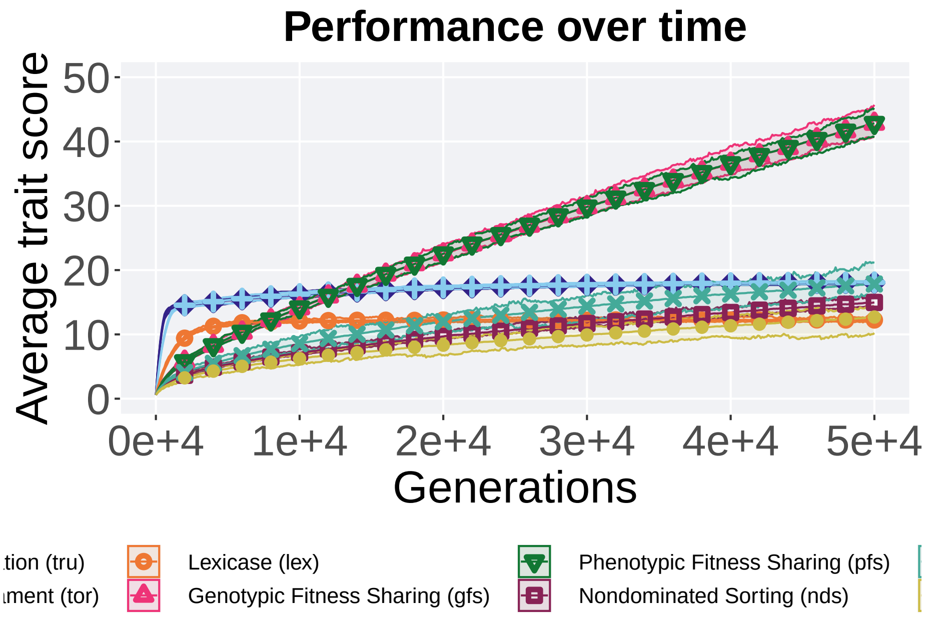
2.5.1 Performance over time

Best performance in a population over time.

```
# data for lines and shading on plots
lines = filter(cc_over_time_mvc, diagnostic == 'exploitation_rate') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

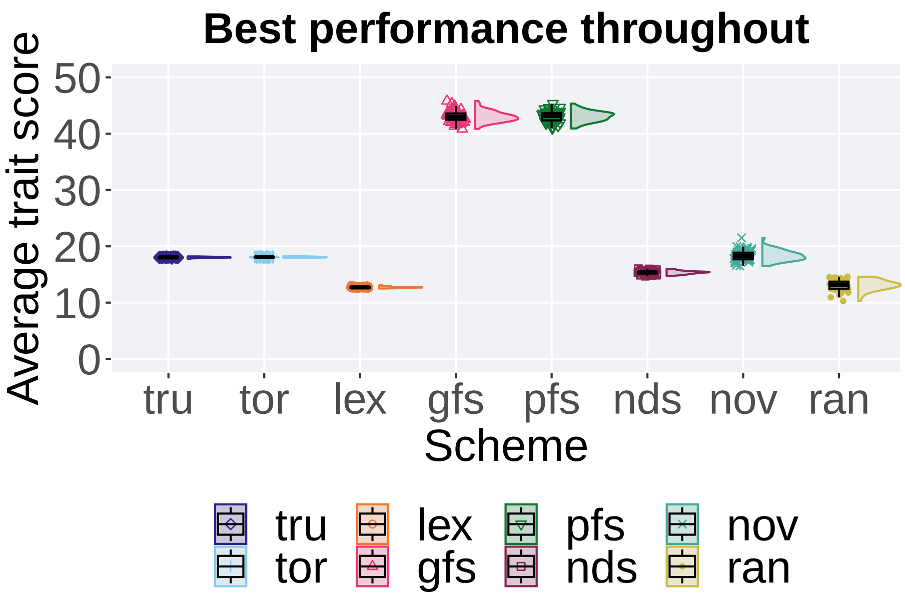
```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 50),
    breaks=seq(0,50, 10),
    labels=c("0", "10", "20", "30", "40", "50")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")

  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Performance over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    sh=guide_legend(ncol=2, title.position = "left"),
    color=guide_legend(ncol=2, title.position = "left"),
    fillape=guide_legend(ncol=2, title.position = "left")
  )
```



2.5.2 Best performance throughout

Best performance found throughout 50,000 generations.

```r
### best performance throughout
filter(cc_best_mvc, col == 'pop_fit_max' & diagnostic == 'exploitation_rate') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha =
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 50),
    breaks=seq(0,50, 10),
    labels=c("0", "10", "20", "30", "40", "50")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best performance throughout')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



2.5.2.1 Stats

Summary statistics for the performance of the best performance.

```
#get data & summarize
performance = filter(cc_best_mvc, col == 'pop_fit_max' & diagnostic == 'exploitation_rate')
performance$acron = factor(performance$acron, levels = c('gfs','pfs','tru','tor','nov', 'nds','le
performance %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val / DIMENSIONALITY, na.rm = TRUE),
    median = median(val / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(val / DIMENSIONALITY, na.rm = TRUE),
    max = max(val / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(val / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 gfs      50      0  40.8   43.0  43.0  45.8 1.12
## 2 pfs      50      0  40.9   43.1  43.1  45.3 1.30
## 3 tru      50      0  17.8   18.0  18.0  18.2 0.118
## 4 tor      50      0  17.9   18.1  18.1  18.3 0.130
## 5 nov      50      0  16.5   18.3  18.3  21.5 1.19
## 6 nds      50      0  14.7   15.4  15.3  16.0 0.318
## 7 lex      50      0  12.5   12.7  12.7  13.1 0.121
## 8 ran      50      0  10.3   13.2  13.1  14.6 1.25
```

Kruskal–Wallis test provides evidence of statistical differences.

```
kruskal.test(val ~ acron, data = performance)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 366.01, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction.

```
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bon
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

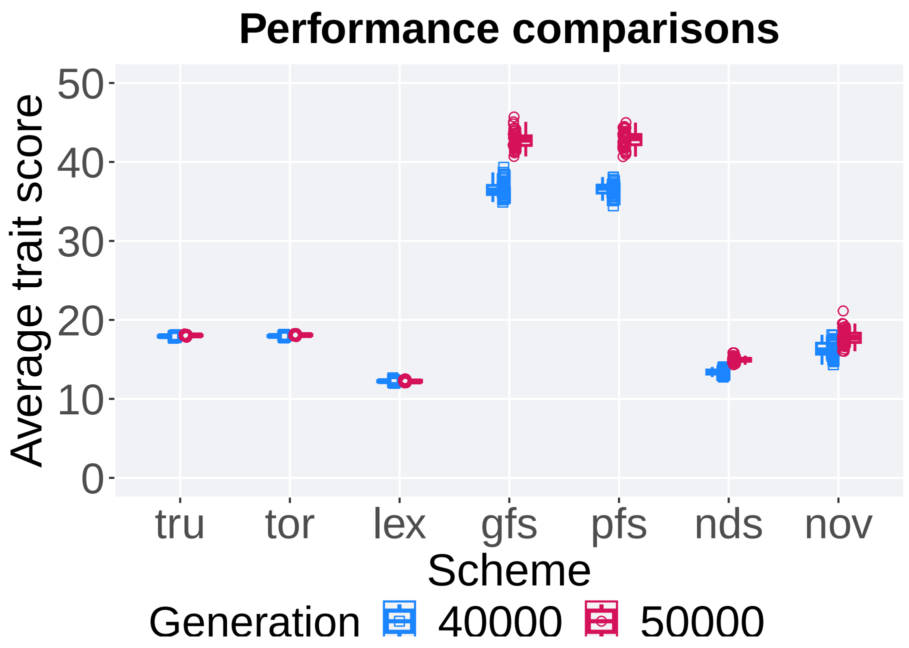
```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  performance$val and performance$acron
##
##     gfs    pfs    tru    tor    nov    nds    lex
## pfs 1      -      -      -      -      -      -
## tru <2e-16 <2e-16 -      -      -      -      -
## tor <2e-16 <2e-16 1      -      -      -      -
## nov <2e-16 <2e-16 1      1      -      -      -
## nds <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -      -
## lex <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -
## ran <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 1
##
## P value adjustment method: bonferroni
```

2.5.3 Performance comparison

Best performances in the population at 40,000 and 50,000 generations.

```
## Warning: The following aesthetics were dropped during statistical transformation:
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
## The following aesthetics were dropped during statistical transformation:
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```

```
# 80% and final generation comparison
end = filter(cc_over_time_mvc, diagnostic == 'exploitation_rate' & gen == 50000 & acron != 'ran')
end$Generation <- factor(end$gen)

mid = filter(cc_over_time_mvc, diagnostic == 'exploitation_rate' & gen == 40000 & acron != 'ran')
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = acron, y=pop_fit_max / DIMENSIONALITY, group = acron, shape = Generat
          geom_point(col = mvc_col[1]  , position = position_jitternudge(jitter.width = .03, nudge
          geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], f

          geom_point(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2]
          geom_boxplot(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), position = pos

          scale_y_continuous(
          name="Average trait score",
          limits=c(0, 50),
          breaks=seq(0,50, 10),
          labels=c("0", "10", "20", "30", "40", "50")
          ) +
          scale_x_discrete(
          name="Scheme"
          )+
          scale_shape_manual(values=c(0,1))+
          scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
          p_theme

plot_grid(
        mvc_p +
        ggtitle("Performance comparisons") +
        theme(legend.position="none"),
        legend,
        nrow=2,
        rel_heights = c(1,.05),
        label_size = TSIZE
      )
```



2.5.3.1 Stats

Summary statistics for the performance of the best performance at 40,000 and 50,000 generations.

```
### performance comparisons and generation slices 40K & 50K
slices = filter(cc_over_time_mvc, diagnostic == 'exploitation_rate' & (gen == 50000 | g
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices$acron = factor(slices$acron, levels = c('gfs','pfs','tru','tor','nov', 'nds','le
slices %>%
  group_by(acron, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max  / DIMENSIONALITY)),
    min = min(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    median = median(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    max = max(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'acron'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 9
## # Groups:   acron [7]
##    acron Generation count na_cnt   min median  mean   max   IQR
##    <fct> <fct>      <int>  <int> <dbl>  <dbl> <dbl> <dbl> <dbl>
##  1 gfs   50000         50      0  40.7   42.8  42.8  45.7 1.21
##  2 gfs   40000         50      0  34.9   36.4  36.6  39.3 1.15
##  3 pfs   50000         50      0  40.7   43.0  42.8  45.0 1.30
##  4 pfs   40000         50      0  34.4   36.7  36.6  38.1 1.01
##  5 tru   50000         50      0  17.8   18.0  18.0  18.2 0.118
##  6 tru   40000         50      0  17.7   17.9  17.9  18.1 0.147
##  7 tor   50000         50      0  17.9   18.1  18.1  18.3 0.130
##  8 tor   40000         50      0  17.7   18.0  18.0  18.2 0.115
##  9 nov   50000         50      0  16.0   17.8  17.8  21.1 1.17
## 10 nov   40000         50      0  14.3   16.1  16.3  18.1 1.39
## 11 nds   50000         50      0  14.3   15.0  15.0  15.8 0.327
## 12 nds   40000         50      0  12.8   13.4  13.4  14.0 0.516
## 13 lex   50000         50      0  12.0   12.2  12.2  12.5 0.199
## 14 lex   40000         50      0  12.0   12.2  12.2  12.7 0.132
```

Truncation selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tru' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'tru' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tru" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 2037.5, p-value = 5.705e-08
## alternative hypothesis: true location shift is not equal to 0
```

Tournament selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tor' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'tor' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tor" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 2075, p-value = 1.301e-08
## alternative hypothesis: true location shift is not equal to 0
```

Lexicase selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'lex' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'lex' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
```

```
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "lex" & Generation == 50000)$pop_fit_max and filter(s
## W = 1260.5, p-value = 0.945
## alternative hypothesis: true location shift is not equal to 0
```

Genotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'gfs' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'gfs' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "gfs" & Generation == 50000)$pop_fit_max and filter(s
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

Phenotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'pfs' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'pfs' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "pfs" & Generation == 50000)$pop_fit_max and filter(s
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

Nondominated sorting comparisons.

```
wilcox.test(x = filter(slices, acron == 'nds' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'nds' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nds" & Generation == 50000)$pop_fit_max and filter(s
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

Novelty search comparisons.

```
wilcox.test(x = filter(slices, acron == 'nov' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'nov' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nov" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 2196, p-value = 7.119e-11
## alternative hypothesis: true location shift is not equal to 0
```

Chapter 3

Ordered exploitation results

Here we present the results for **best performances** found by each selection scheme replicate on the ordered exploitation diagnostic. Best performance found refers to the largest average trait score found in a given population. Note that performance values fall between 0.0 and 100.0.

3.1 Analysis dependencies

```r
library(ggplot2)
library(cowplot)
library(dplyr)
library(PupillometryR)
library(sdamr)
```

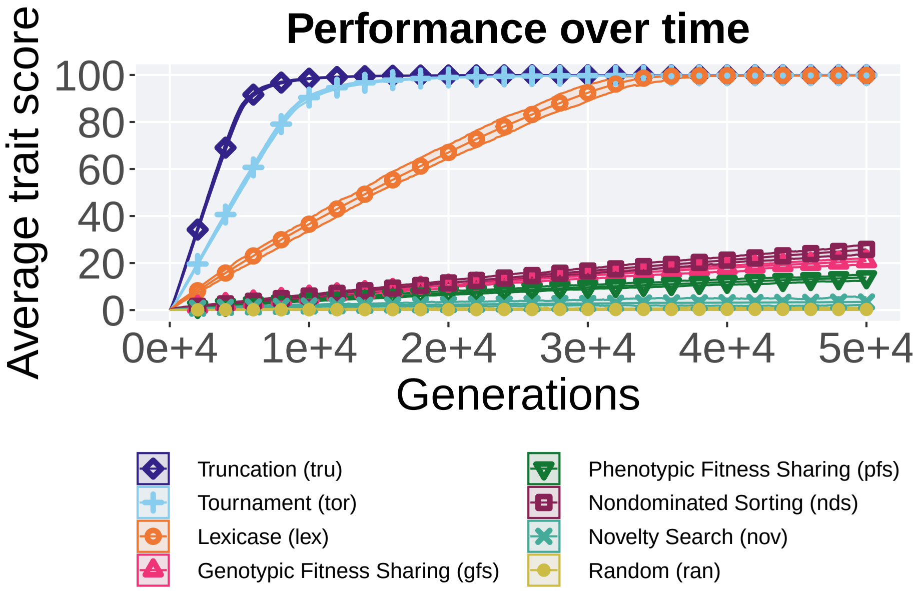
3.2 Performance over time

Best performance in a population over time.

```r
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'ordered_exploitation') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

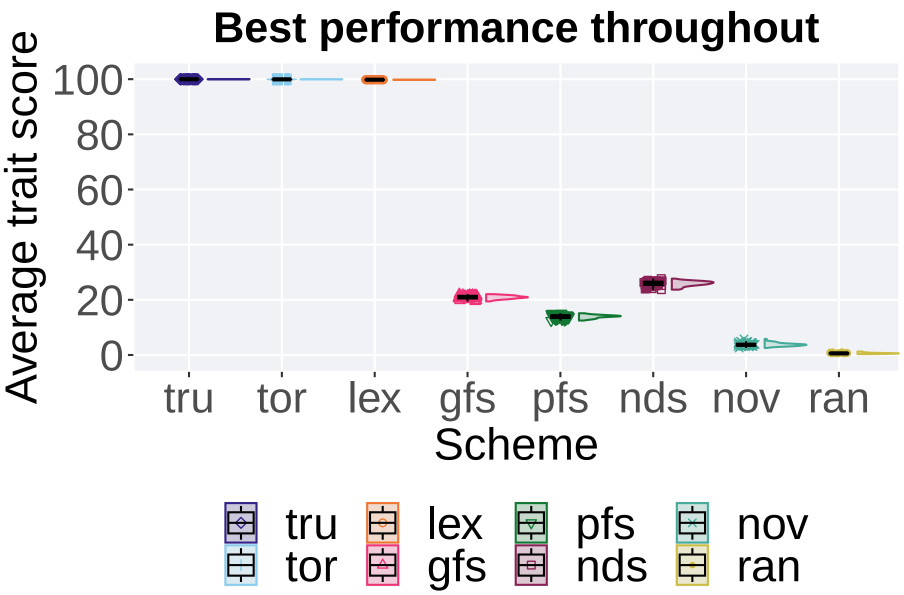
```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")

  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Performance over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```



3.3 Best performance throughout

Best performance found throughout 50,000 generations.

```
### best performance throughout
filter(cc_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape = acron))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Average trait score",
    limits=c(-1, 101),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best performance throughout')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



3.3.1 Stats

Summary statistics for the performance of the best performance throughout 50,000 generations.

```
#get data & summarize
performance = filter(cc_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitati
performance$acron = factor(performance$acron, levels = c('tru', 'tor', 'lex','nds', 'g
performance %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val / DIMENSIONALITY, na.rm = TRUE),
    median = median(val / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(val / DIMENSIONALITY, na.rm = TRUE),
    max = max(val / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(val / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt     min  median    mean    max     IQR
##   <fct> <int>  <int>   <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
## 1 tru      50      0 100.    100.    100.    100.   0.00208
## 2 tor      50      0  99.9    99.9    99.9    99.9  0.00445
## 3 lex      50      0  99.8    99.8    99.8    99.8  0.0207
## 4 nds      50      0  23.7    26.0    25.9    27.7  1.17
## 5 gfs      50      0  19.4    21.0    20.9    22.1  0.970
## 6 pfs      50      0  12.5    14.1    13.9    15.1  0.871
## 7 nov      50      0   2.55    3.70    3.80    5.82 0.718
## 8 ran      50      0   0.319   0.598   0.634   1.26 0.240
```

Kruskal–Wallis test provides evidence of statistical differences.

```r
kruskal.test(val ~ acron, data = performance)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 392.77, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction.

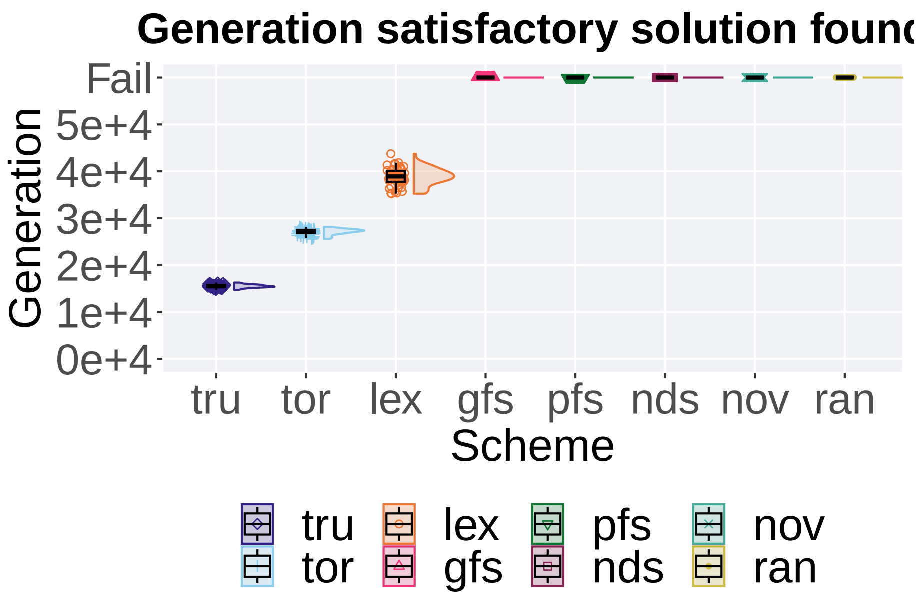
```r
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bonferroni",
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  performance$val and performance$acron
##
##     tru    tor    lex    nds    gfs    pfs    nov
## tor <2e-16 -      -      -      -      -      -
## lex <2e-16 <2e-16 -      -      -      -      -
## nds <2e-16 <2e-16 <2e-16 -      -      -      -
## gfs <2e-16 <2e-16 <2e-16 <2e-16 -      -      -
## pfs <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -      -
## nov <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -
## ran <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

3.4 Generation satisfactory solution found

First generation a satisfactory solution is found throughout the 50,000 generations.

```
### satisfactory solution found
filter(cc_ssf, diagnostic == 'ordered_exploitation') %>%
  ggplot(., aes(x = acron, y = Generations , color = acron, fill = acron, shape = acron
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Generation",
    limits=c(0, 60001),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000, 60000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4", "Fail")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Generation satisfactory solution found')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



3.4.1 Stats

Summary statistics for the first generation a satisfactory solution is found throughout the 50,000 generations.

```
### Generation satisfactory solution found
ssf = filter(cc_ssf, diagnostic == 'ordered_exploitation'  & Generations < 60000)
ssf$acron = factor(ssf$acron, levels = c('tru', 'tor', 'lex'))
ssf %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(Generations)),
    min = min(Generations, na.rm = TRUE),
    median = median(Generations, na.rm = TRUE),
    mean = mean(Generations, na.rm = TRUE),
    max = max(Generations, na.rm = TRUE),
    IQR = IQR(Generations, na.rm = TRUE)
  )
```

```
## # A tibble: 3 x 8
##   acron count na_cnt   min median   mean   max   IQR
##   <fct> <int>  <int> <int>  <dbl>  <dbl> <int> <dbl>
## 1 tru      50      0 14701 15466. 15511. 16280  422.
## 2 tor      50      0 25563 27254. 27122. 28151  714
## 3 lex      50      0 35240 38918. 38865. 43751 2316.
```

Kruskal–Wallis test provides evidence of difference amoung selection schemes.

```
kruskal.test(Generations ~ acron, data = ssf)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Generations by acron
## Kruskal-Wallis chi-squared = 132.45, df = 2, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction.

```
pairwise.wilcox.test(x = ssf$Generations, g = ssf$acron, p.adjust.method = "bonferroni
                     paired = FALSE, conf.int = FALSE, alternative = 'g')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  ssf$Generations and ssf$acron
##
##     tru    tor
## tor <2e-16 -
## lex <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

3.5 Multi-valley crossing results

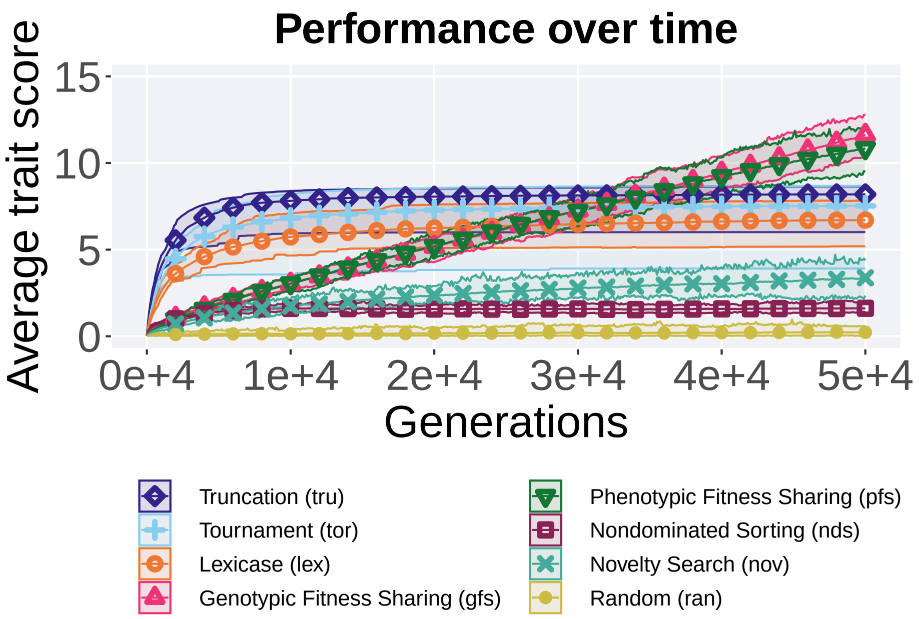
3.5.1 Performance over time

Best performance in a population over time.

```
# data for lines and shading on plots
lines = filter(cc_over_time_mvc, diagnostic == 'ordered_exploitation') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

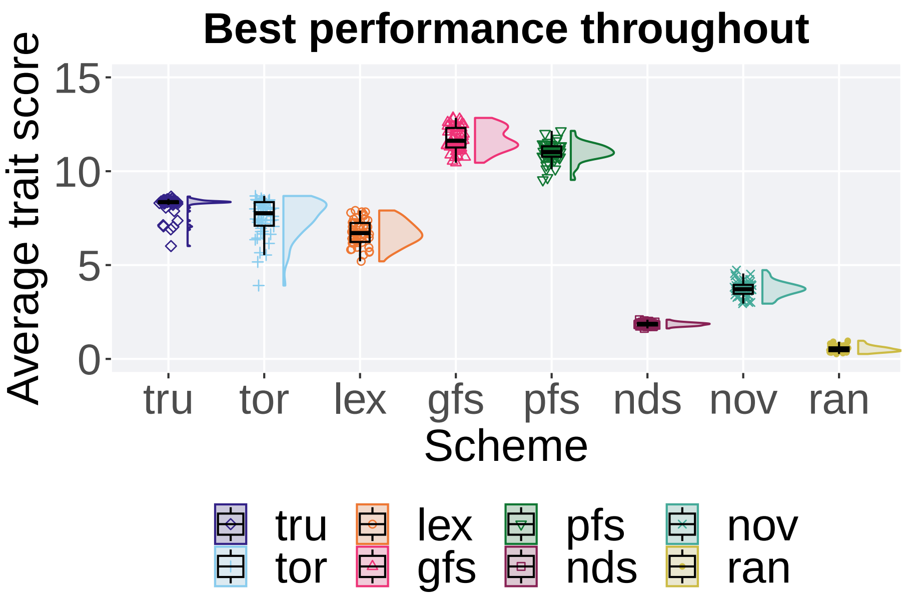
```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 15),
    breaks=seq(0,15, 5),
    labels=c("0", "5", "10", "15")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")

  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Performance over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "left"),
    color=guide_legend(ncol=2, title.position = "left"),
    fill=guide_legend(ncol=2, title.position = "left")
  )
```



3.5.2 Best performance throughout

Best performance found throughout 50,000 generations.

```r
### best performance throughout
filter(cc_best_mvc, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  guides(fill = "none",color = 'none', shape = 'none') +
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 15),
    breaks=seq(0,15, 5),
    labels=c("0", "5", "10", "15")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best performance throughout')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



3.5.2.1 Stats

Summary statistics for the performance of the best performance.

```
#get data & summarize
performance = filter(cc_best_mvc, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation')
performance$acron = factor(performance$acron, levels = c('gfs','pfs','tru','tor','lex','nov', 'nd
performance %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val / DIMENSIONALITY, na.rm = TRUE),
    median = median(val / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(val / DIMENSIONALITY, na.rm = TRUE),
    max = max(val / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(val / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max    IQR
##   <fct> <int>  <int> <dbl>  <dbl> <dbl> <dbl>  <dbl>
## 1 gfs      50      0 10.5   11.6  11.7  12.8  1.04
## 2 pfs      50      0  9.54  11.0  11.0  12.1  0.553
## 3 tru      50      0  6.01   8.35  8.19  8.65 0.0922
## 4 tor      50      0  3.91   7.76  7.52  8.68 1.26
## 5 lex      50      0  5.20   6.70   6.72   7.91  1.01
## 6 nov      50      0  2.95   3.71   3.72   4.73  0.476
## 7 nds      50      0  1.63   1.86   1.85   2.09  0.129
## 8 ran      50      0  0.263  0.490  0.534  0.968 0.202
```

Kruskal–Wallis test provides evidence of statistical differences.

```
kruskal.test(val ~ acron, data = performance)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 380.23, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction.

```
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bon
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

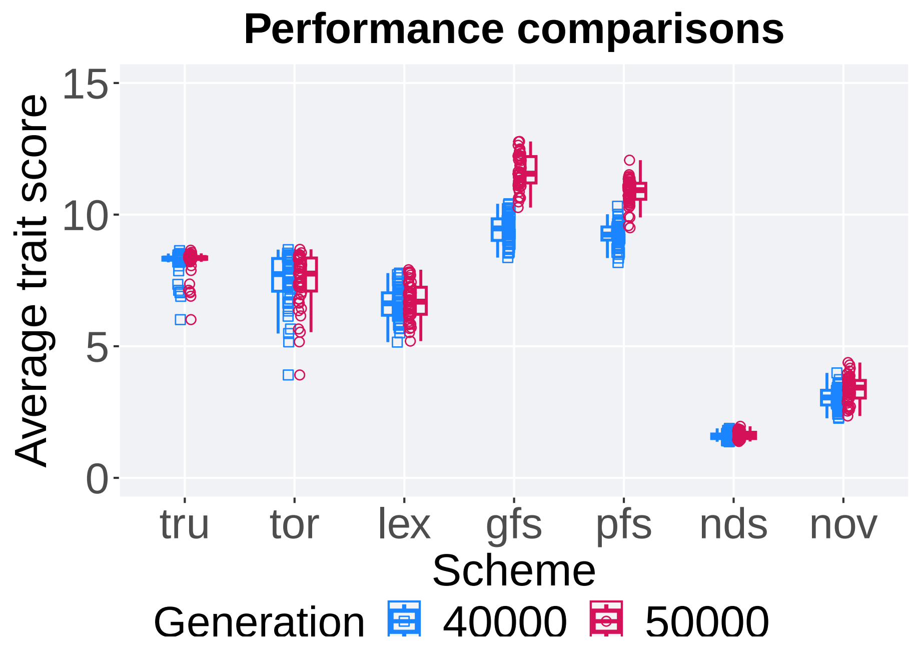
```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  performance$val and performance$acron
##
##     gfs     pfs     tru     tor     lex     nov     nds
## pfs 1.6e-06 -       -       -       -       -       -
## tru < 2e-16 < 2e-16 -       -       -       -       -
## tor < 2e-16 < 2e-16 0.0026  -       -       -       -
## lex < 2e-16 < 2e-16 7.7e-14 1.7e-05 -       -       -
## nov < 2e-16 < 2e-16 < 2e-16 2.4e-16 < 2e-16 -       -
## nds < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
## ran < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16
##
## P value adjustment method: bonferroni
```

3.5.3 Performance comparison

Best performances in the population at 40,000 and 50,000 generations.

```
## Warning: The following aesthetics were dropped during statistical transformation:
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
## The following aesthetics were dropped during statistical transformation:
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```

```
# 80% and final generation comparison
end = filter(cc_over_time_mvc, diagnostic == 'ordered_exploitation' & gen == 50000 & acron != 'ra
end$Generation <- factor(end$gen)

mid = filter(cc_over_time_mvc, diagnostic == 'ordered_exploitation' & gen == 40000 & acron != 'ra
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = acron, y=pop_fit_max / DIMENSIONALITY, group = acron, shape = Generat
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mv

  geom_point(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], positi
  geom_boxplot(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), position = position_nu

  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 15),
    breaks=seq(0,15, 5),
    labels=c("0", "5", "10", "15")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=c(0,1))+
  scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
  p_theme

plot_grid(
  mvc_p +
    ggtitle("Performance comparisons") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)
```



3.5.3.1 Stats

Summary statistics for the performance of the best performance at 40,000 and 50,000 generations.

```
### performance comparisons and generation slices 40K & 50K
slices = filter(cc_over_time_mvc, diagnostic == 'ordered_exploitation' & (gen == 50000
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices$acron = factor(slices$acron, levels = c('gfs','pfs','tru','tor','lex','nov', 'n
slices %>%
  group_by(acron, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max  / DIMENSIONALITY)),
    min = min(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    median = median(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    max = max(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'acron'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 9
## # Groups:   acron [7]
##    acron Generation count na_cnt   min median  mean   max    IQR
##    <fct> <fct>      <int>  <int> <dbl>  <dbl> <dbl> <dbl>  <dbl>
##  1 gfs   50000         50      0 10.3   11.6  11.6  12.8  1.00
##  2 gfs   40000         50      0  8.37   9.48  9.45 10.4  0.820
##  3 pfs   50000         50      0  9.50  10.9  10.8  12.1  0.606
##  4 pfs   40000         50      0  8.18   9.24  9.23 10.3  0.498
##  5 tru   50000         50      0  6.01   8.35  8.19  8.65 0.0922
##  6 tru   40000         50      0  6.01   8.33  8.17  8.63 0.112
##  7 tor   50000         50      0  3.91   7.76  7.52  8.68 1.26
##  8 tor   40000         50      0  3.91   7.74  7.49  8.67 1.24
##  9 lex   50000         50      0  5.19   6.69  6.70  7.91 1.03
## 10 lex   40000         50      0  5.16   6.63  6.63  7.78 0.852
## 11 nov   50000         50      0  2.35   3.43  3.38  4.38 0.670
## 12 nov   40000         50      0  2.27   3.06  3.03  3.99 0.560
## 13 nds   50000         50      0  1.38   1.63  1.61  1.96 0.239
## 14 nds   40000         50      0  1.37   1.58  1.58  1.88 0.173
```

Truncation selection comparisons.

```r
wilcox.test(x = filter(slices, acron == 'tru' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'tru' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tru" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 1375, p-value = 0.3907
## alternative hypothesis: true location shift is not equal to 0
```

Tournament selection comparisons.

```r
wilcox.test(x = filter(slices, acron == 'tor' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'tor' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tor" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 1306.5, p-value = 0.6995
## alternative hypothesis: true location shift is not equal to 0
```

Lexicase selection comparisons.

```r
wilcox.test(x = filter(slices, acron == 'lex' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'lex' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
```

```
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "lex" & Generation == 50000)$pop_fit_max and filter(s
## W = 1348, p-value = 0.5015
## alternative hypothesis: true location shift is not equal to 0
```

Genotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'gfs' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'gfs' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "gfs" & Generation == 50000)$pop_fit_max and filter(s
## W = 2498, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

Phenotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'pfs' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'pfs' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "pfs" & Generation == 50000)$pop_fit_max and filter(s
## W = 2471, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

Nondominated sorting comparisons.

```
wilcox.test(x = filter(slices, acron == 'nds' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'nds' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nds" & Generation == 50000)$pop_fit_max and filter(s
## W = 1413, p-value = 0.2626
## alternative hypothesis: true location shift is not equal to 0
```

Novelty search comparisons.

```
wilcox.test(x = filter(slices, acron == 'nov' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'nov' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nov" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 1789, p-value = 0.0002054
## alternative hypothesis: true location shift is not equal to 0
```

Chapter 4

Contradictory objectives results

Here we present the results for the **satisfactory trait corverage** and **activation gene coverage** generated by each selection scheme replicate on the contradictory objectives diagnostic. Note both of these values are gathered at the population-level. Activation gene coverage refers to the count of unique activation genes in a given population; this gives us a range of integers between 0 and 100. Satisfactory trait coverage refers to the count of unique satisfied traits in a given population; this gives us a range of integers between 0 and 100.

4.1 Analysis dependencies

```
library(ggplot2)
library(cowplot)
library(dplyr)
library(PupillometryR)
library(sdamr)
```

4.2 Satisfactory trait coverage

Satisfactory trait coverage analysis.

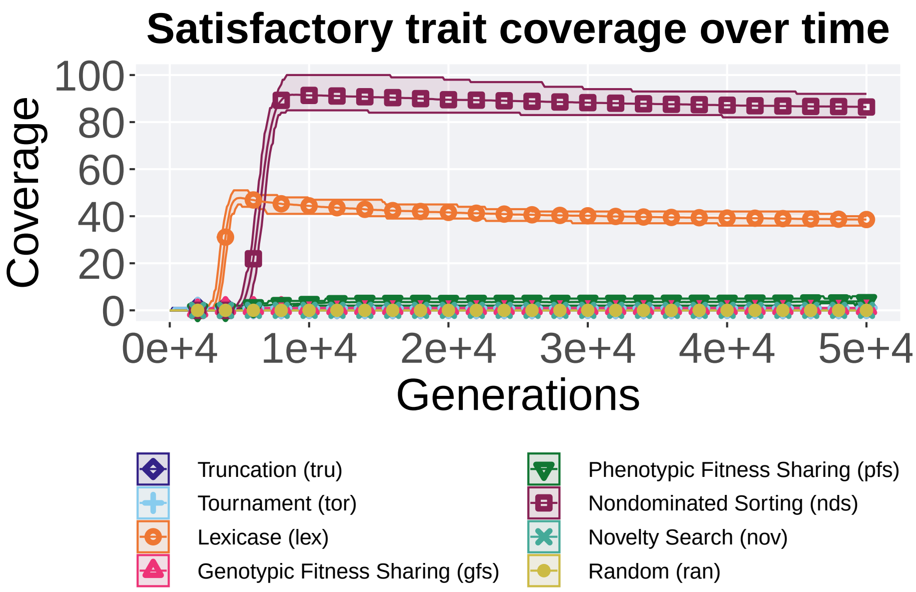
4.2.1 Coverage over time

Satisfactory trait coverage over time.

```r
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'contradictory_objectives') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

```r
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Satisfactory trait coverage over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```

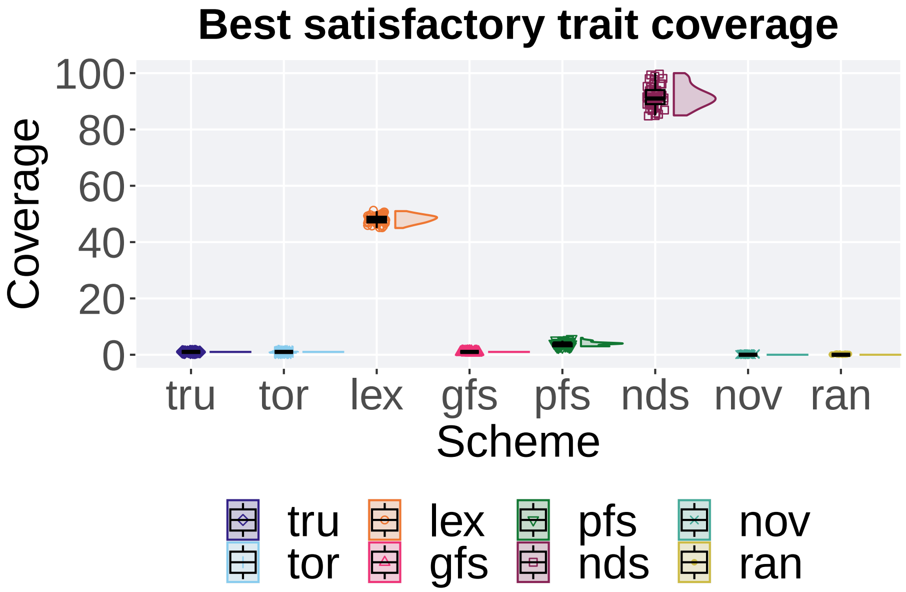


4.2.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
### best satisfactory trait coverage throughout
filter(cc_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = acron, y = val, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best satisfactory trait coverage')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```

```
## Warning: Removed 50 rows containing missing values (`geom_point()`).
```



4.2.2.1 Stats

Summary statistics for the best satisfactory trait coverage.

```
### best
coverage = filter(cc_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objecti
coverage$acron = factor(coverage$acron, levels = c('nds', 'lex', 'pfs', 'gfs', 'tor',
coverage %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val, na.rm = TRUE),
    median = median(val, na.rm = TRUE),
    mean = mean(val, na.rm = TRUE),
    max = max(val, na.rm = TRUE),
    IQR = IQR(val, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 nds      50      0    85     91 91.8    100     5
## 2 lex      50      0    45     48 48.2     51     2
## 3 pfs      50      0     3      4  3.84     6     1
## 4 gfs      50      0     1      1  1        1     0
## 5 tor      50      0     1      1  1        1     0
## 6 tru      50      0     1      1  1        1     0
## 7 nov      50      0     0      0  0        0     0
## 8 ran      50      0     0      0  0        0     0
```

Kruskal–Wallis test provides evidence of difference among satisfactory trait coverage.

```r
kruskal.test(val ~ acron, data = coverage)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 396.67, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on satisfactory trait coverage.

```r
pairwise.wilcox.test(x = coverage$val, g = coverage$acron, p.adjust.method = "bonferroni",
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

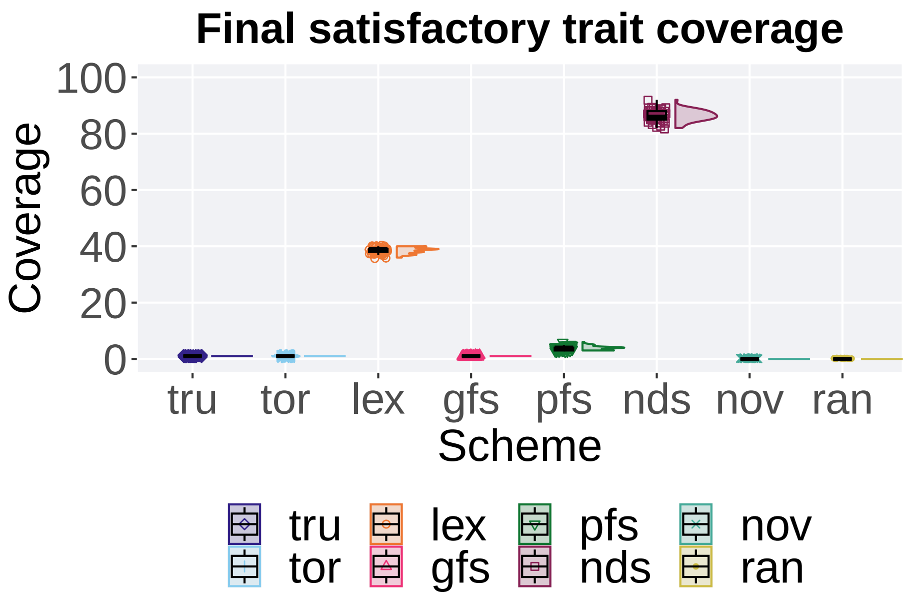
```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$val and coverage$acron
##
##     nds    lex    pfs    gfs    tor    tru    nov
## lex <2e-16 -      -      -      -      -      -
## pfs <2e-16 <2e-16 -      -      -      -      -
## gfs <2e-16 <2e-16 <2e-16 -      -      -      -
## tor <2e-16 <2e-16 <2e-16 1      -      -      -
## tru <2e-16 <2e-16 <2e-16 1      1      -      -
## nov <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -
## ran <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 1
##
## P value adjustment method: bonferroni
```

4.2.3 End of 50,000 generations

Satisfactory trait coverage in the population at the end of 50,000 generations.

```
### end of run
filter(cc_over_time, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = acron, y = pop_uni_obj, color = acron, fill = acron, shape = acron
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Final satisfactory trait coverage')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```

```
## Warning: Removed 50 rows containing missing values (`geom_point()`).
```



4.2.3.1 Stats

Summary statistics for satisfactory trait coverage in the population at the end of 50,000 generations.

```
### end of run
coverage = filter(cc_over_time, diagnostic == 'contradictory_objectives' & gen == 50000)
coverage$acron = factor(coverage$acron, levels = c('nds', 'lex', 'pfs', 'gfs', 'tor', 'tru', 'nov
coverage %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_uni_obj)),
    min = min(pop_uni_obj, na.rm = TRUE),
    median = median(pop_uni_obj, na.rm = TRUE),
    mean = mean(pop_uni_obj, na.rm = TRUE),
    max = max(pop_uni_obj, na.rm = TRUE),
    IQR = IQR(pop_uni_obj, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
## 1 nds      50      0    82     86 86.4     92     3
## 2 lex      50      0    36     39 38.6     40     1
## 3 pfs      50      0     3      4  3.82     6     1
## 4 gfs       50       0      1       1  1         1     0
## 5 tor       50       0      1       1  1         1     0
## 6 tru       50       0      1       1  1         1     0
## 7 nov       50       0      0       0  0         0     0
## 8 ran       50       0      0       0  0         0     0
```

Kruskal–Wallis test provides evidence of difference among satisfactory trait coverage in the population at the end of 50,000 generations.

```
kruskal.test(pop_uni_obj ~ acron, data = coverage)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  pop_uni_obj by acron
## Kruskal-Wallis chi-squared = 396.7, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on satisfactory trait coverage in the population at the end of 50,000 generations.

```
pairwise.wilcox.test(x = coverage$pop_uni_obj, g = coverage$acron, p.adjust.method = "
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$pop_uni_obj and coverage$acron
##
##     nds    lex    pfs    gfs    tor    tru    nov
## lex <2e-16 -      -      -      -      -      -
## pfs <2e-16 <2e-16 -      -      -      -      -
## gfs <2e-16 <2e-16 <2e-16 -      -      -      -
## tor <2e-16 <2e-16 <2e-16 1      -      -      -
## tru <2e-16 <2e-16 <2e-16 1      1      -      -
## nov <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -
## ran <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 1
##
## P value adjustment method: bonferroni
```

4.3 Activation gene coverage

Activation gene coverage analysis.

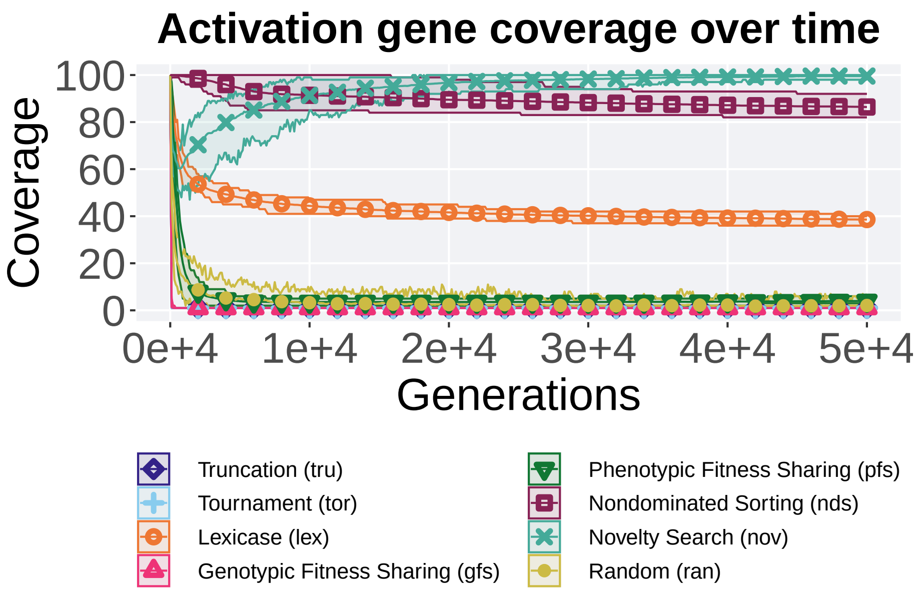
4.3.1 Over time coverage

Activation gene coverage over time.

```r
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'contradictory_objectives') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

```r
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme`, color =
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Activation gene coverage over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```

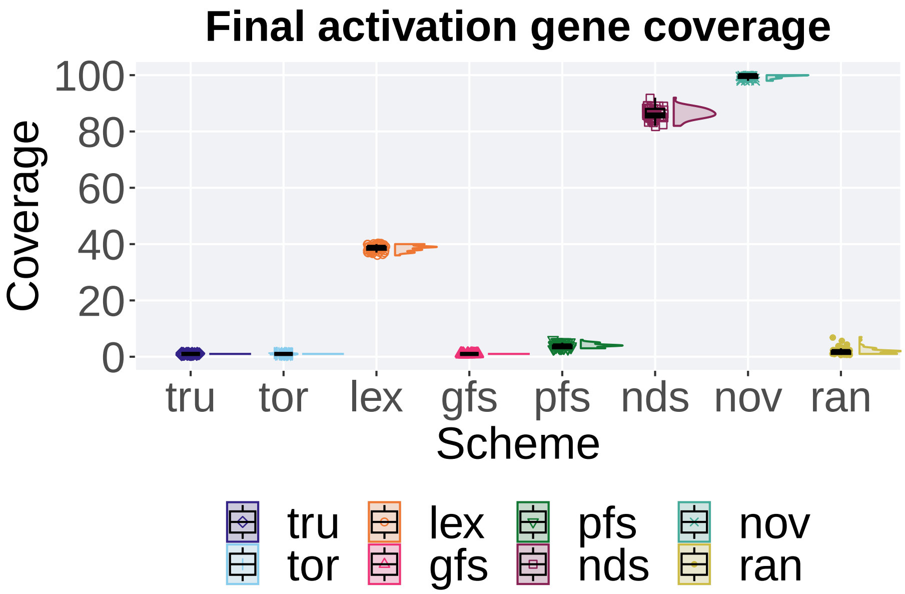


4.3.2 End of 50,000 generations

Activation gene coverage in the population at the end of 50,000 generations.

```
# end of run
filter(cc_over_time, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = acron, y = uni_str_pos, color = acron, fill = acron, shape = acron)
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Final activation gene coverage')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```

```
## Warning: Removed 19 rows containing missing values (`geom_point()`).
```



4.3.2.1 Stats

Summary statistics for activation gene coverage.

```
# end of run
coverage = filter(cc_over_time, diagnostic == 'contradictory_objectives' & gen == 50000)
coverage$acron = factor(coverage$acron, levels = c('nov', 'nds', 'lex', 'pfs', 'ran', 'gfs', 'tor
coverage %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(uni_str_pos)),
    min = min(uni_str_pos, na.rm = TRUE),
    median = median(uni_str_pos, na.rm = TRUE),
    mean = mean(uni_str_pos, na.rm = TRUE),
    max = max(uni_str_pos, na.rm = TRUE),
    IQR = IQR(uni_str_pos, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
## 1 nov      50      0    98    100 99.6    100     1
## 2 nds      50      0    82     86 86.4     92     3
## 3 lex      50      0    36     39 38.6     40     1
## 4 pfs      50      0     3      4  3.98     6     1
## 5 ran      50      0     1      2  2.06     7     1
## 6 gfs      50      0     1      1  1        1     0
## 7 tor      50      0     1      1  1        1     0
## 8 tru      50      0     1      1  1        1     0
```

Kruskal–Wallis test provides evidence of difference among activation gene coverage.

```
kruskal.test(uni_str_pos ~ acron, data = coverage)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  uni_str_pos by acron
## Kruskal-Wallis chi-squared = 384.23, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on activation gene coverage.

```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$acron, p.adjust.method = "
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$uni_str_pos and coverage$acron
##
##     nov     nds     lex     pfs     ran     gfs tor
## nds < 2e-16 -       -       -       -       -   -
## lex < 2e-16 < 2e-16 -       -       -       -   -
## pfs < 2e-16 < 2e-16 < 2e-16 -       -       -   -
## ran < 2e-16 < 2e-16 < 2e-16 2.9e-12 -       -   -
## gfs < 2e-16 < 2e-16 < 2e-16 < 2e-16 3.0e-10 -   -
## tor < 2e-16 < 2e-16 < 2e-16 < 2e-16 3.0e-10 1   -
## tru < 2e-16 < 2e-16 < 2e-16 < 2e-16 3.0e-10 1   1
##
## P value adjustment method: bonferroni
```

4.4 Nondominated sorting split

Here analyze the satisfactory trait coverage and activation gene coverage results for nondominated sorting, nondominated front ranking (no fitness sharing between fronts), and phenotypic fitness sharing.

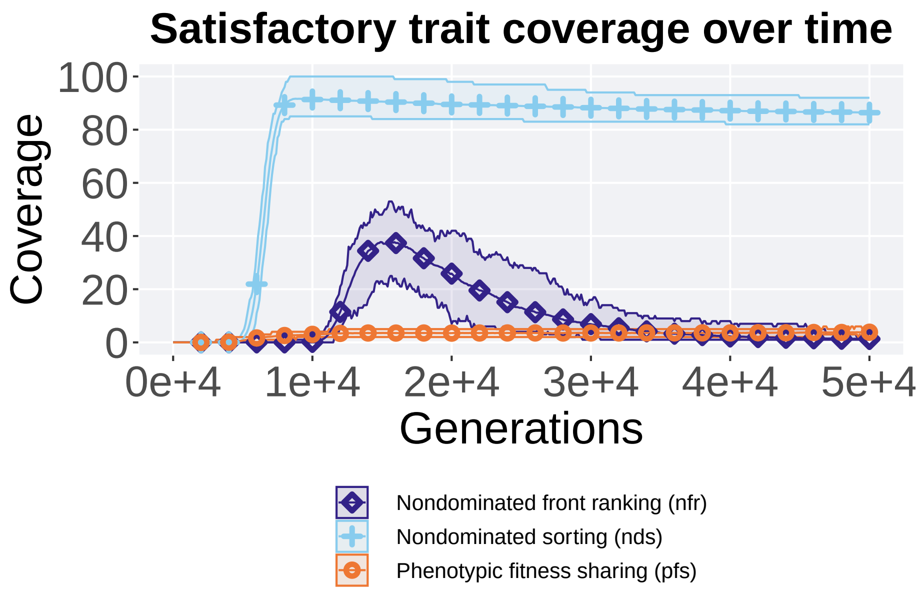
4.4.1 Coverage over time

Satisfactory trait coverage over time.

```
lines = filter(nss, diagnostic == 'contradictory_objectives') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme`, color =
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Satisfactory trait coverage over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=1, title.position = "bottom"),
    color=guide_legend(ncol=1, title.position = "bottom"),
    fill=guide_legend(ncol=1, title.position = "bottom")
  )
```

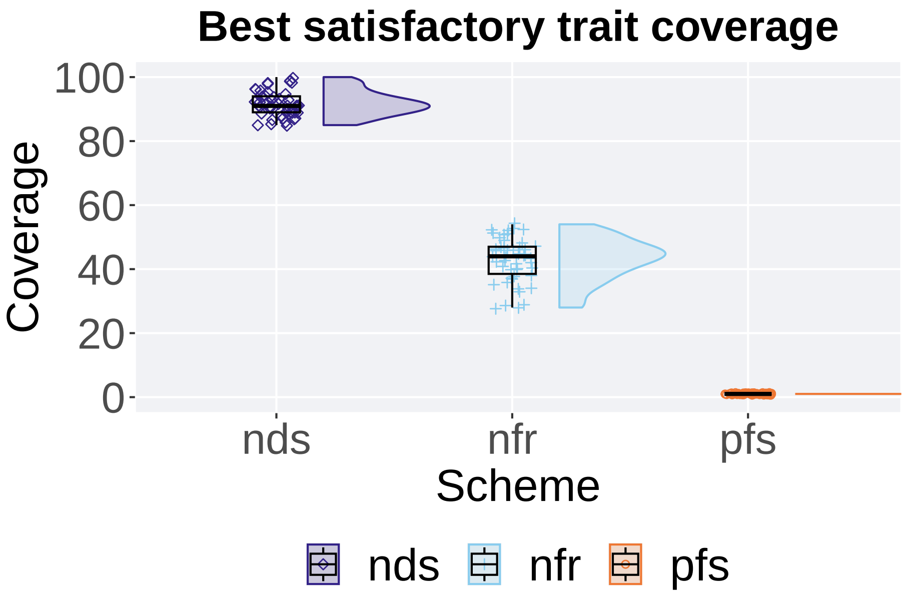


4.4.2 Best coverage throughout

Best satisfactory trait coverage.

```
### best satisfactory trait coverage throughout
coverage %>%
  ggplot(., aes(x = acron, y = val, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha =
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best satisfactory trait coverage')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=1, title.position = "bottom"),
    color=guide_legend(nrow=1, title.position = "bottom"),
    fill=guide_legend(nrow=1, title.position = "bottom")
  )
```

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
```



4.4.2.1 Stats

Summary statistics for the best satisfactory trait coverage.

```
# summary
coverage$acron = factor(coverage$acron, levels = c('nds', 'pfs', 'nfr'))
coverage %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val, na.rm = TRUE),
    median = median(val, na.rm = TRUE),
    mean = mean(val, na.rm = TRUE),
    max = max(val, na.rm = TRUE),
    IQR = IQR(val, na.rm = TRUE)
  )
```

```
## # A tibble: 3 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 nds      50      0    85     91  91.8   100   5
## 2 pfs      50      0     1      1   1       1   0
## 3 nfr      50      0    28     44  42.8    54   8.5
```

Kruskal–Wallis test provides evidence of difference among best satisfactory trait coverage.

```
kruskal.test(val ~ acron,data = coverage)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 137.61, df = 2, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on best satisfactory trait coverage.

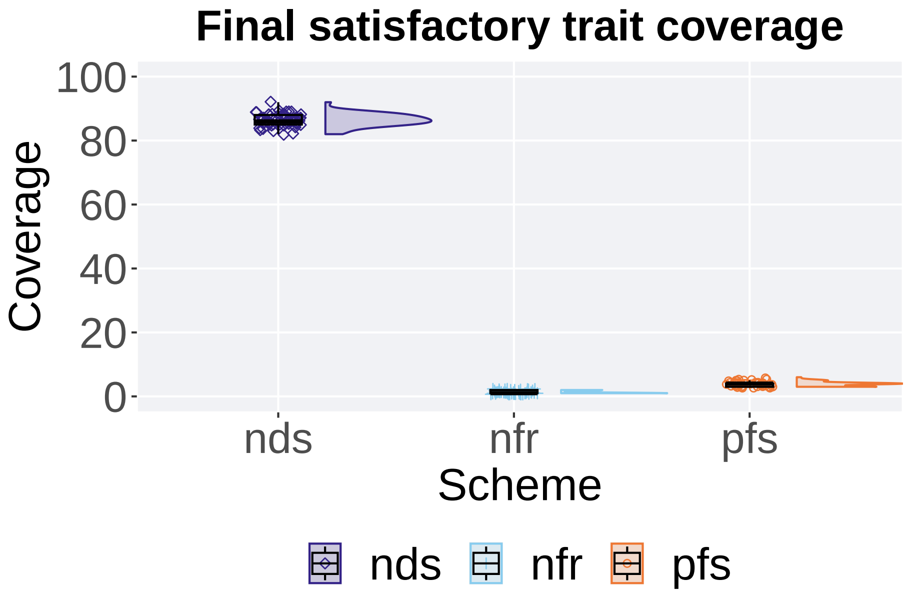
```
pairwise.wilcox.test(x = coverage$val, g = coverage$acron, p.adjust.method = "bonferro
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$val and coverage$acron
##
##     nds    pfs
## pfs <2e-16 -
## nfr <2e-16 1
##
## P value adjustment method: bonferroni
```

4.4.3 End of 50,000 generations

Satisfactory trait coverage in the population at the end of 50,000 generations.

```
coverage %>%
  ggplot(., aes(x = acron, y = pop_uni_obj, color = acron, fill = acron, shape = acron
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Final satisfactory trait coverage')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=1, title.position = "bottom"),
    color=guide_legend(nrow=1, title.position = "bottom"),
    fill=guide_legend(nrow=1, title.position = "bottom")
  )
```



4.4.3.1 Stats

Summary statistics for satisfactory trait coverage in the population at the end of 50,000 generations.

```
coverage$acron = factor(coverage$acron, levels = c('nds', 'pfs', 'nfr'))
coverage %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_uni_obj)),
    min = min(pop_uni_obj, na.rm = TRUE),
    median = median(pop_uni_obj, na.rm = TRUE),
    mean = mean(pop_uni_obj, na.rm = TRUE),
    max = max(pop_uni_obj, na.rm = TRUE),
    IQR = IQR(pop_uni_obj, na.rm = TRUE)
  )
```

```
## # A tibble: 3 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
## 1 nds      50      0    82     86 86.4     92     3
## 2 pfs      50      0     3      4  3.82     6     1
## 3 nfr      50      0     1      1  1.28     2     1
```

Kruskal–Wallis test provides evidence of difference among satisfactory trait coverage in the population at the end of 50,000 generations.

```
kruskal.test(pop_uni_obj ~ acron,data = coverage)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  pop_uni_obj by acron
## Kruskal-Wallis chi-squared = 135.36, df = 2, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on satisfactory trait coverage in the population at the end of 50,000 generations.

```
pairwise.wilcox.test(x = coverage$pop_uni_obj, g = coverage$acron, p.adjust.method = "
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$pop_uni_obj and coverage$acron
##
##     nds    pfs
## pfs <2e-16 -
## nfr <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

4.5 Multi-valley crossing results

4.5.1 Satisfactory trait coverage

Satisfactory trait coverage analysis.

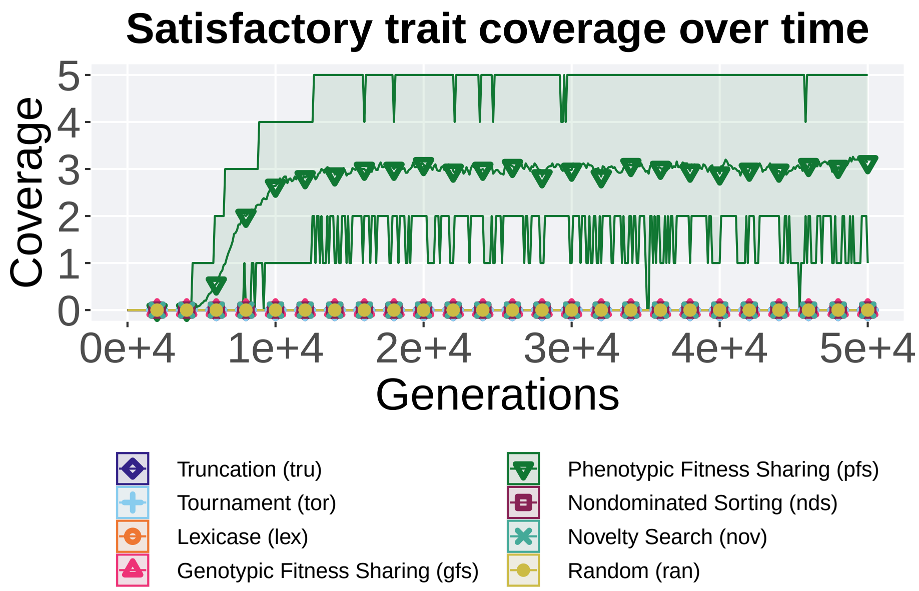
4.5.1.1 Coverage over time

Satisfactory trait coverage over time.

```r
# data for lines and shading on plots
lines = filter(cc_over_time_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

```r
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme`, color =
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 5),
    breaks=seq(0,5, 1)
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Satisfactory trait coverage over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```

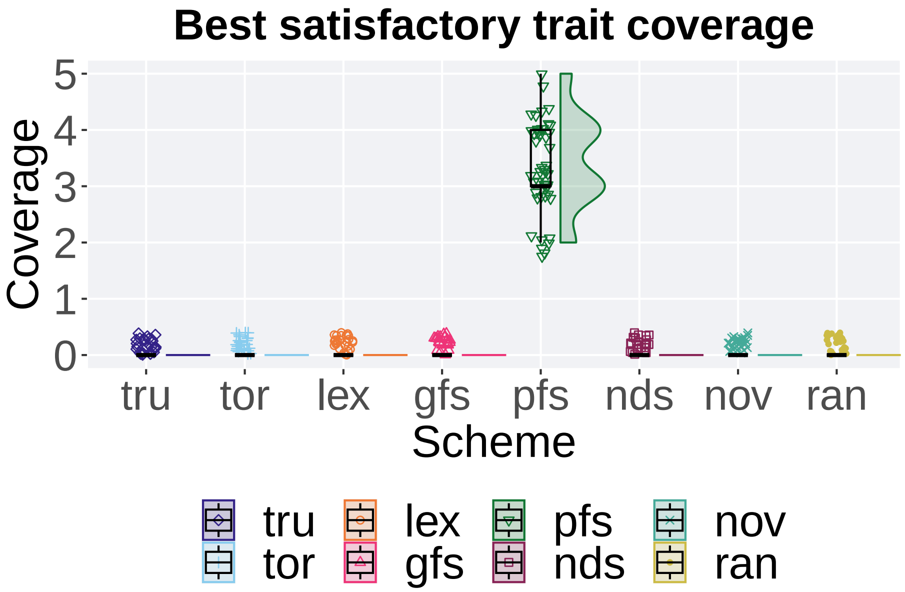


4.5.1.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
### best satisfactory trait coverage throughout
filter(cc_best_mvc, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>
  ggplot(., aes(x = acron, y = val, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  guides(fill = "none",color = 'none', shape = 'none') +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 5)
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best satisfactory trait coverage')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```

```
## Warning: Removed 168 rows containing missing values (`geom_point()`).
```



4.5.1.2.1 Stats

Summary statistics for the best satisfactory trait coverage.

```
### best
coverage = filter(cc_best_mvc, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives')
coverage$acron = factor(coverage$acron, levels = c('pfs','nds', 'lex',  'gfs', 'tor', 'tru', 'nov
coverage %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val, na.rm = TRUE),
    median = median(val, na.rm = TRUE),
    mean = mean(val, na.rm = TRUE),
    max = max(val, na.rm = TRUE),
    IQR = IQR(val, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 pfs      50      0     2      3  3.42     5     1
## 2 nds      50      0     0      0  0        0     0
## 3 lex      50      0     0      0  0        0     0
## 4 gfs      50      0     0      0  0        0     0
## 5 tor      50      0     0      0  0        0     0
## 6 tru      50      0     0      0  0        0     0
## 7 nov      50      0     0      0  0        0     0
## 8 ran      50      0     0      0  0        0     0
```

Kruskal–Wallis test provides evidence of difference among satisfactory trait coverage.

```
kruskal.test(val ~ acron, data = coverage)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 396.91, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on satisfactory trait coverage.

```
pairwise.wilcox.test(x = coverage$val, g = coverage$acron, p.adjust.method = "bonferro
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$val and coverage$acron
##
##     pfs    nds lex gfs tor tru nov
## nds <2e-16 -   -   -   -   -   -
## lex <2e-16 1   -   -   -   -   -
## gfs <2e-16 1   1   -   -   -   -
## tor <2e-16 1   1   1   -   -   -
## tru <2e-16 1   1   1   1   -   -
## nov <2e-16 1   1   1   1   1   -
## ran <2e-16 1   1   1   1   1   1
##
## P value adjustment method: bonferroni
```

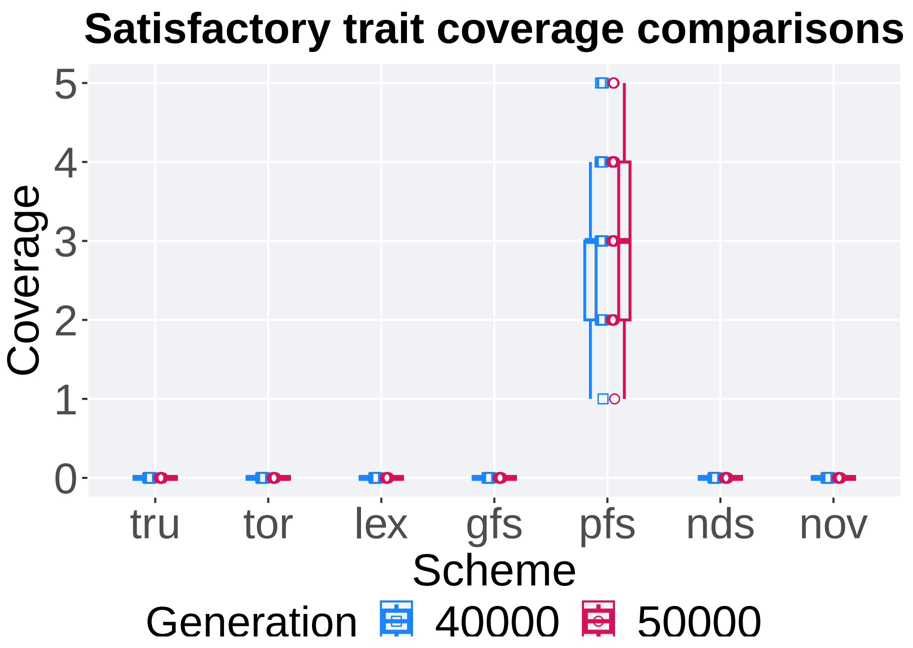
4.5.1.3 Coverage comparison

Best performances in the population at 40,000 and 50,000 generations.

```
## Warning: The following aesthetics were dropped during statistical transformation:
```

```
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
## The following aesthetics were dropped during statistical transformation:
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```

```
end = filter(cc_over_time_mvc, diagnostic == 'contradictory_objectives' & gen == 50000 & acron !=
end$Generation <- factor(end$gen)

mid = filter(cc_over_time_mvc, diagnostic == 'contradictory_objectives' & gen == 40000 & acron !=
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = acron, y=pop_uni_obj, group = acron, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mv

  geom_point(data = end, aes(x = acron, y=pop_uni_obj), col = mvc_col[2], position = position_jit
  geom_boxplot(data = end, aes(x = acron, y=pop_uni_obj), position = position_nudge(x = .15, y =

  scale_y_continuous(
    name="Coverage",
    limits=c(0, 5)
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=c(0,1))+
  scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
  p_theme

plot_grid(
  mvc_p +
    ggtitle("Satisfactory trait coverage comparisons") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)
```



4.5.1.3.1 Stats

Summary statistics for the activation gene coverage at 40,000 and 50,000 generations.

```
slices = filter(cc_over_time_mvc, diagnostic == 'contradictory_objectives' & (gen == 50
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices$acron = factor(slices$acron, levels = c('pfs','nds', 'lex',  'gfs', 'tor', 'tru
slices %>%
  group_by(acron, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_uni_obj)),
    min = min(pop_uni_obj, na.rm = TRUE),
    median = median(pop_uni_obj, na.rm = TRUE),
    mean = mean(pop_uni_obj, na.rm = TRUE),
    max = max(pop_uni_obj, na.rm = TRUE),
    IQR = IQR(pop_uni_obj, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'acron'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 9
## # Groups:   acron [7]
##    acron Generation count na_cnt   min median  mean   max   IQR
##    <fct> <fct>      <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
##  1 pfs   50000         50      0     1      3  3.14     5     2
##  2 pfs   40000         50      0     1      3  2.9      5     1
##  3 nds   50000         50      0     0      0  0        0     0
##  4 nds   40000         50      0     0      0  0        0     0
##  5 lex   50000         50      0     0      0  0        0     0
##  6 lex   40000         50      0     0      0  0        0     0
##  7 gfs   50000         50      0     0      0  0        0     0
##  8 gfs   40000         50      0     0      0  0        0     0
##  9 tor   50000         50      0     0      0  0        0     0
## 10 tor   40000         50      0     0      0  0        0     0
## 11 tru   50000         50      0     0      0  0        0     0
## 12 tru   40000         50      0     0      0  0        0     0
## 13 nov   50000         50      0     0      0  0        0     0
## 14 nov   40000         50      0     0      0  0        0     0
```

Truncation selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tru' & Generation == 50000)$pop_uni_obj,
            y = filter(slices, acron == 'tru' & Generation == 40000)$pop_uni_obj,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tru" & Generation == 50000)$pop_uni_obj and filter(slices, acr
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

Tournament selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tor' & Generation == 50000)$pop_uni_obj,
            y = filter(slices, acron == 'tor' & Generation == 40000)$pop_uni_obj,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tor" & Generation == 50000)$pop_uni_obj and filter(slices, acr
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

Lexicase selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'lex' & Generation == 50000)$pop_uni_obj,
            y = filter(slices, acron == 'lex' & Generation == 40000)$pop_uni_obj,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
```

```
##
## data:  filter(slices, acron == "lex" & Generation == 50000)$pop_uni_obj and filter(
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

Genotypic fitness sharing comparisons.

```r
wilcox.test(x = filter(slices, acron == 'gfs' & Generation == 50000)$pop_uni_obj,
            y = filter(slices, acron == 'gfs' & Generation == 40000)$pop_uni_obj,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "gfs" & Generation == 50000)$pop_uni_obj and filter(
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

Phenotypic fitness sharing comparisons.

```r
wilcox.test(x = filter(slices, acron == 'pfs' & Generation == 50000)$pop_uni_obj,
            y = filter(slices, acron == 'pfs' & Generation == 40000)$pop_uni_obj,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "pfs" & Generation == 50000)$pop_uni_obj and filter(
## W = 1423.5, p-value = 0.2118
## alternative hypothesis: true location shift is not equal to 0
```

Nondominated sorting comparisons.

```r
wilcox.test(x = filter(slices, acron == 'nds' & Generation == 50000)$pop_uni_obj,
            y = filter(slices, acron == 'nds' & Generation == 40000)$pop_uni_obj,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nds" & Generation == 50000)$pop_uni_obj and filter(
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

Novelty search comparisons.

```r
wilcox.test(x = filter(slices, acron == 'nov' & Generation == 50000)$pop_uni_obj,
            y = filter(slices, acron == 'nov' & Generation == 40000)$pop_uni_obj,
            alternative = 't')
```

```
##
```

```
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nov" & Generation == 50000)$pop_uni_obj and filter(slices, acr
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

4.5.2 Activation gene coverage

Activation gene coverage analysis.

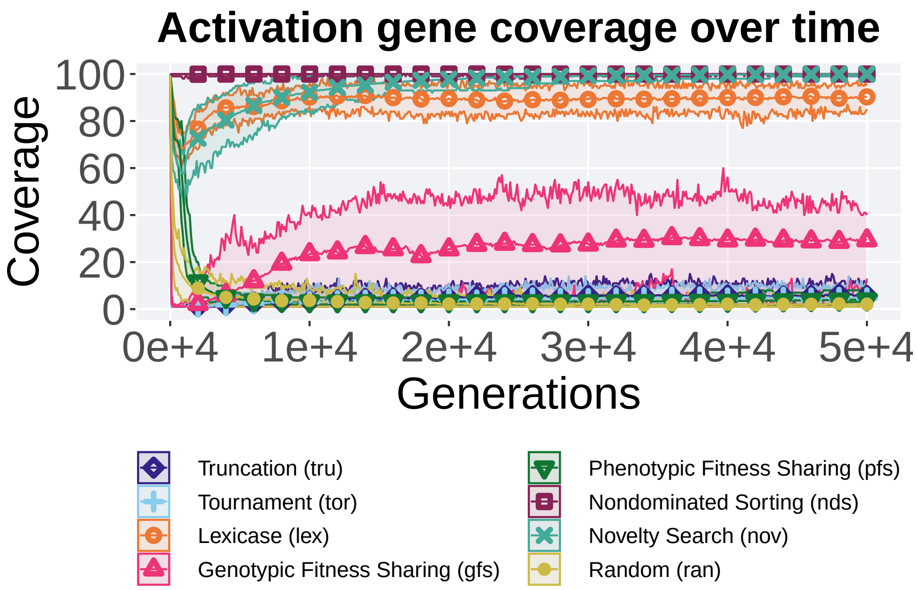
4.5.2.1 Coverage over time

Activation gene coverage over time.

```
lines = filter(cc_over_time_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

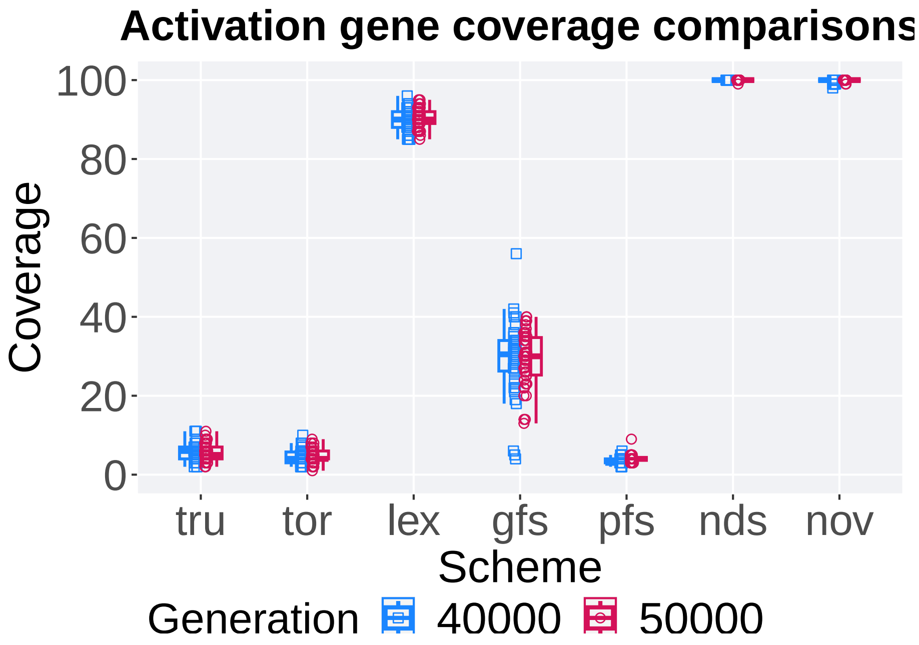
```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme`, color =
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Activation gene coverage over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```



4.5.2.2 Coverage comparison

Best activation gene coverage in the population at 40,000 and 50,000 generations.

```
mvc_p = ggplot(mid, aes(x = acron, y=uni_str_pos, group = acron, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nuc
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1]

  geom_point(data = end, aes(x = acron, y=uni_str_pos), col = mvc_col[2], position = po
  geom_boxplot(data = end, aes(x = acron, y=uni_str_pos), position = position_nudge(x =

  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=c(0,1))+
  scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
  p_theme

plot_grid(
  mvc_p +
    ggtitle("Activation gene coverage comparisons") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)
```



4.5.2.3 Stats

Summary statistics for the activation gene coverage at 40,000 and 50,000 generations.

```
slices = filter(cc_over_time_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices$acron = factor(slices$acron, levels = c('nov','nds','lex','gfs','tor','tru','pfs'))
slices %>%
  group_by(acron, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(uni_str_pos)),
    min = min(uni_str_pos, na.rm = TRUE),
    median = median(uni_str_pos, na.rm = TRUE),
    mean = mean(uni_str_pos, na.rm = TRUE),
    max = max(uni_str_pos, na.rm = TRUE),
    IQR = IQR(uni_str_pos, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'acron'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 9
## # Groups:   acron [7]
##    acron Generation count na_cnt   min median   mean   max   IQR
##    <fct> <fct>      <int>  <int> <int>  <dbl>  <dbl> <int> <dbl>
##  1 nov   50000         50      0    99  100   100.      100  0
##  2 nov   40000         50      0    98  100    99.8     100  0
##  3 nds   50000         50      0    99  100   100.      100  0
##  4 nds   40000         50      0   100  100   100       100  0
##  5 lex   50000         50      0    85   90    90.3      95  3
##  6 lex   40000         50      0    85   90    90.0      96  4
##  7 gfs   50000         50      0    13   30    29.4      40  9.5
##  8 gfs   40000         50      0     4   30.5  29.3      56  7.75
##  9 tor   50000         50      0     1    4     4.64      9  2
## 10 tor   40000         50      0     2    4     4.54     10  2.75
## 11 tru   50000         50      0     2    5     5.6      11  3
## 12 tru   40000         50      0     2    6     5.8      11  3
## 13 pfs   50000         50      0     3    4     4.02      9  0
## 14 pfs   40000         50      0     2    3     3.42      6  1
```

Truncation selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tru' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'tru' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tru" & Generation == 50000)$uni_str_pos and filter(s
## W = 1175, p-value = 0.6039
## alternative hypothesis: true location shift is not equal to 0
```

Tournament selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tor' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'tor' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tor" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 1351.5, p-value = 0.4781
## alternative hypothesis: true location shift is not equal to 0
```

Lexicase selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'lex' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'lex' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "lex" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 1321.5, p-value = 0.6221
## alternative hypothesis: true location shift is not equal to 0
```

Genotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'gfs' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'gfs' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "gfs" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 1223.5, p-value = 0.8575
## alternative hypothesis: true location shift is not equal to 0
```

Phenotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'pfs' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'pfs' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "pfs" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 1709, p-value = 0.0006733
## alternative hypothesis: true location shift is not equal to 0
```

Nondominated sorting comparisons.

```
wilcox.test(x = filter(slices, acron == 'nds' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'nds' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nds" & Generation == 50000)$uni_str_pos and filter(
## W = 1225, p-value = 0.3271
## alternative hypothesis: true location shift is not equal to 0
```

Novelty search comparisons.

```
wilcox.test(x = filter(slices, acron == 'nov' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'nov' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nov" & Generation == 50000)$uni_str_pos and filter(
## W = 1476, p-value = 0.007657
## alternative hypothesis: true location shift is not equal to 0
```

Chapter 5

Multi-path mpeloration results

Here we present the results for the **best performances** and **activation gene coverage** generated by each selection scheme replicate on the multi-path mpeloration diagnostic. Best performance found refers to the largest average trait score found in a given population. Note that activation gene coverage values are gathered at the population-level. Activation gene coverage refers to the count of unique activation genes in a given population; this gives us a range of integers between 0 and 100.

5.1 Analysis dependencies

```
library(ggplot2)
library(cowplot)
library(dplyr)
library(PupillometryR)
library(sdamr)
```

5.2 Performance

Performance analysis.

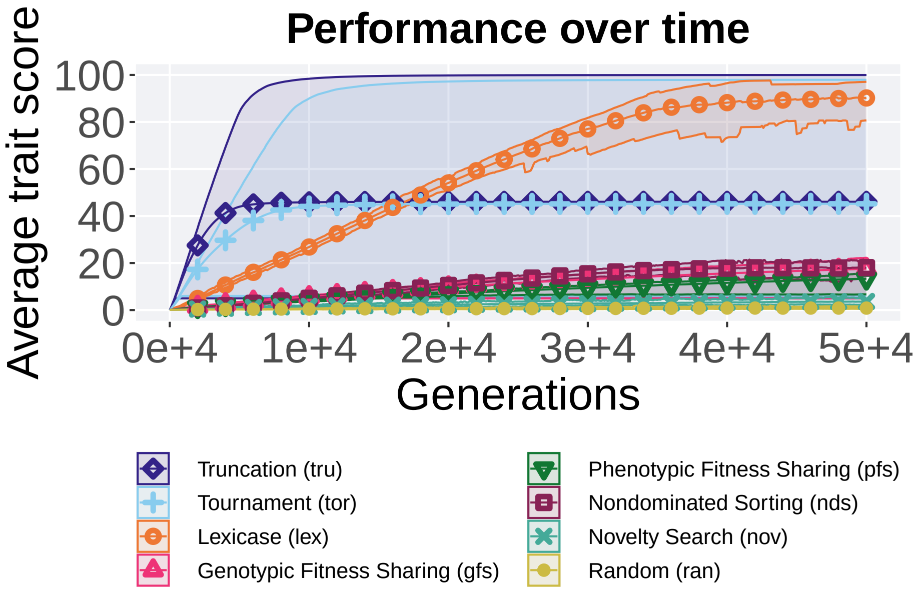
5.2.1 Over time

Best performance in a population over time.

```r
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'multipath_exploration') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

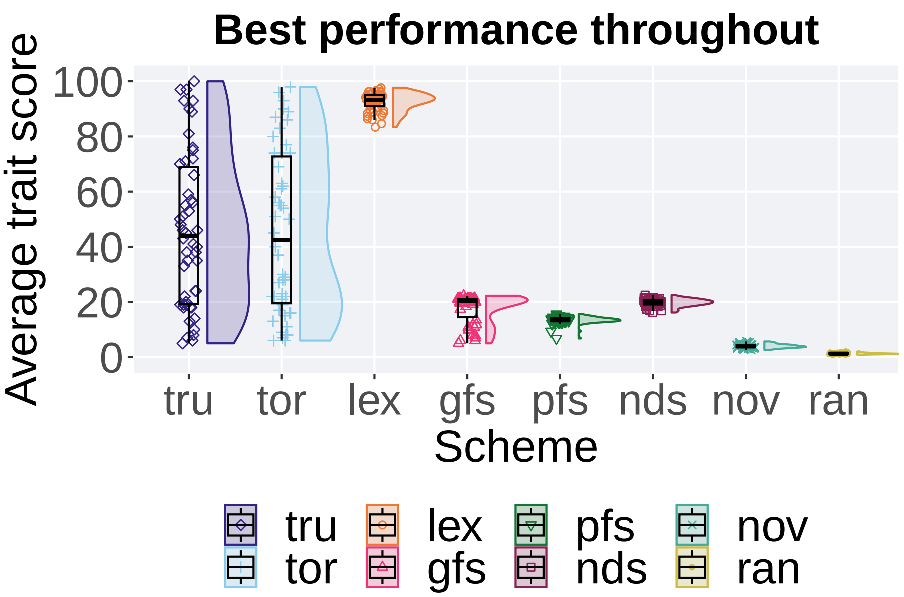
```r
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")

  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Performance over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```



5.2.2 Best performance throughout

Best performance throughout 50,000 generations.

```
### best performance throughout
filter(cc_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape = acron))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  guides(fill = "none",color = 'none', shape = 'none') +
  scale_y_continuous(
    name="Average trait score",
    limits=c(-1, 101),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best performance throughout')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



5.2.2.1 Stats

Summary statistics for the best performance.

```
### best performance throughout
performance = filter(cc_best, col == 'pop_fit_max' & diagnostic == 'multipath_explorat
performance$acron = factor(performance$acron, levels = c('lex','tor','tru','nds','gfs'
performance %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val / DIMENSIONALITY, na.rm = TRUE),
    median = median(val / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(val / DIMENSIONALITY, na.rm = TRUE),
    max = max(val / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(val / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt    min median  mean    max    IQR
##   <fct> <int>  <int>  <dbl>  <dbl> <dbl>  <dbl>  <dbl>
## 1 lex      50      0 83.4    93.2  92.5   97.7   4.05
## 2 tor      50      0  6.00   42.5  45.2   97.9  53.2
## 3 tru      50      0  5      44.0  46.1  100.   49.7
## 4 nds      50      0 16.2    19.9  19.8   22.5   1.59
## 5 gfs      50      0  4.99   20.4  17.6   22.2   6.69
## 6 pfs      50      0  6.76   13.5  13.4   15.6   1.10
## 7 nov      50      0  2.62    3.89  4.01   5.68  0.860
## 8 ran      50      0  0.870   1.25  1.28   2.04  0.288
```

Kruskal–Wallis test provides evidence of difference among best performances.

```r
kruskal.test(val ~ acron, data = performance)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 329.88, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on best performance.

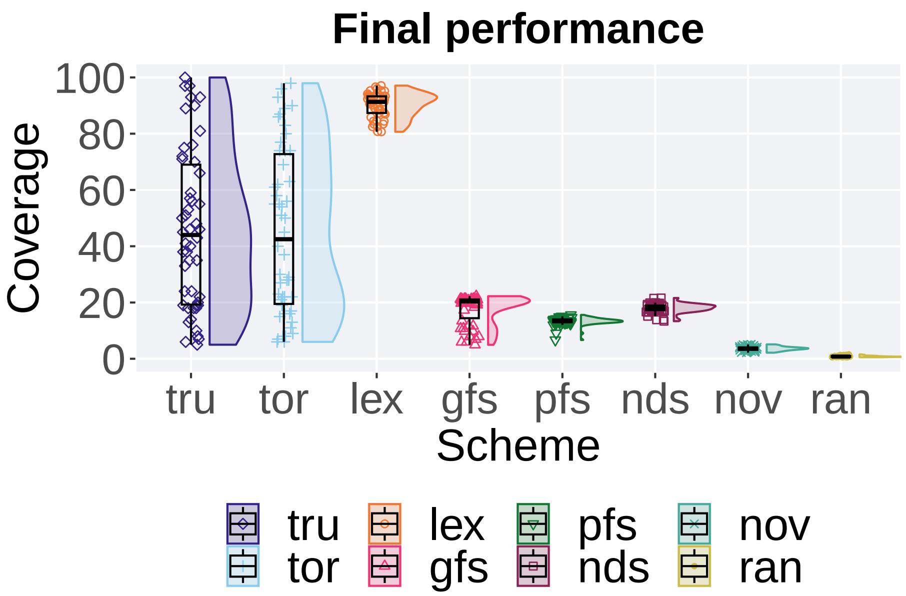
```r
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bonferroni",
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  performance$val and performance$acron
##
##     lex     tor     tru     nds     gfs     pfs     nov
## tor 3.0e-13 -       -       -       -       -       -
## tru 1.1e-11 1.00000 -       -       -       -       -
## nds < 2e-16 0.00047 0.00027 -       -       -       -
## gfs < 2e-16 2.3e-05 1.6e-05 1.00000 -       -       -
## pfs < 2e-16 3.1e-08 6.9e-10 < 2e-16 0.00015 -       -
## nov < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
## ran < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16
##
## P value adjustment method: bonferroni
```

5.2.3 End of 50,000 generations

Best performance in the population at the end of 50,000 generations.

```r
# end of run
filter(cc_over_time, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = acron, y = pop_fit_max / DIMENSIONALITY, color = acron, fill = acro
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  guides(fill = "none",color = 'none', shape = 'none') +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Final performance')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



5.2.3.1 Stats

Summary statistics for best performance in the final population.

```
# end of run
performance = filter(cc_over_time, diagnostic == 'multipath_exploration' & gen == 50000)
performance$acron = factor(performance$acron, levels = c('lex','tor','tru','nds','gfs','pfs','nov
performance %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max / DIMENSIONALITY)),
    min = min(pop_fit_max / DIMENSIONALITY, na.rm = TRUE),
    median = median(pop_fit_max / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(pop_fit_max / DIMENSIONALITY, na.rm = TRUE),
    max = max(pop_fit_max / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(pop_fit_max / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt    min median   mean    max    IQR
##   <fct> <int>  <int>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 lex      50      0 80.7   91.3   90.2    97.1   5.91
## 2 tor      50      0  6.00  42.5   45.2    97.9  53.2
## 3 tru      50      0  5     44.0   46.1   100.   49.7
## 4 nds      50      0 13.4   18.1   18.0    21.6   1.65
## 5 gfs      50      0  4.96  20.4   17.6    22.2   6.68
## 6 pfs      50      0  6.67  13.5   13.3    15.6   1.04
## 7 nov      50      0  2.16   3.66   3.64    5.12  0.859
## 8 ran      50      0  0.553  0.785  0.840   1.56  0.299
```

Kruskal–Wallis test provides evidence of difference among best performance in the final population.

```
kruskal.test(pop_fit_max ~ acron, data = performance)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  pop_fit_max by acron
## Kruskal-Wallis chi-squared = 330.05, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on best performance in the final population.

```
pairwise.wilcox.test(x = performance$pop_fit_max, g = performance$acron, p.adjust.method = "bonfe
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
```

```
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  performance$pop_fit_max and performance$acron
##
##     lex     tor     tru     nds     gfs     pfs     nov
## tor 3.9e-12 -       -       -       -       -       -
## tru 7.1e-11 1.00000 -       -       -       -       -
## nds < 2e-16 8.2e-05 9.3e-07 -       -       -       -
## gfs < 2e-16 2.2e-05 1.6e-05 1.00000 -       -       -
## pfs < 2e-16 3.0e-08 6.6e-10 3.0e-15 0.00015 -       -
## nov < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
## ran < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16
##
## P value adjustment method: bonferroni
```

5.3 Activation gene coverage

Activation gene coverage analysis.

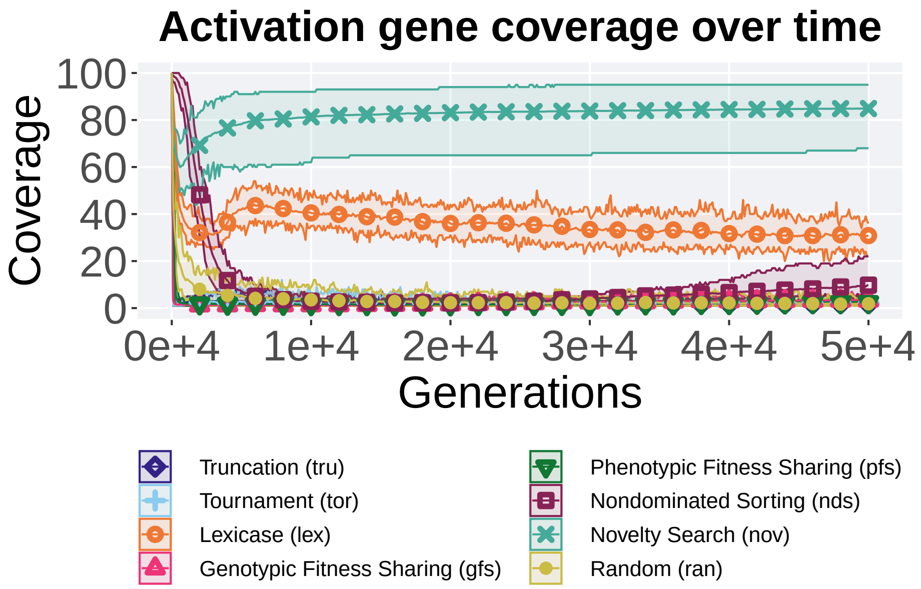
5.3.1 Over time coverage

Activation gene coverage over time.

```
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'multipath_exploration') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

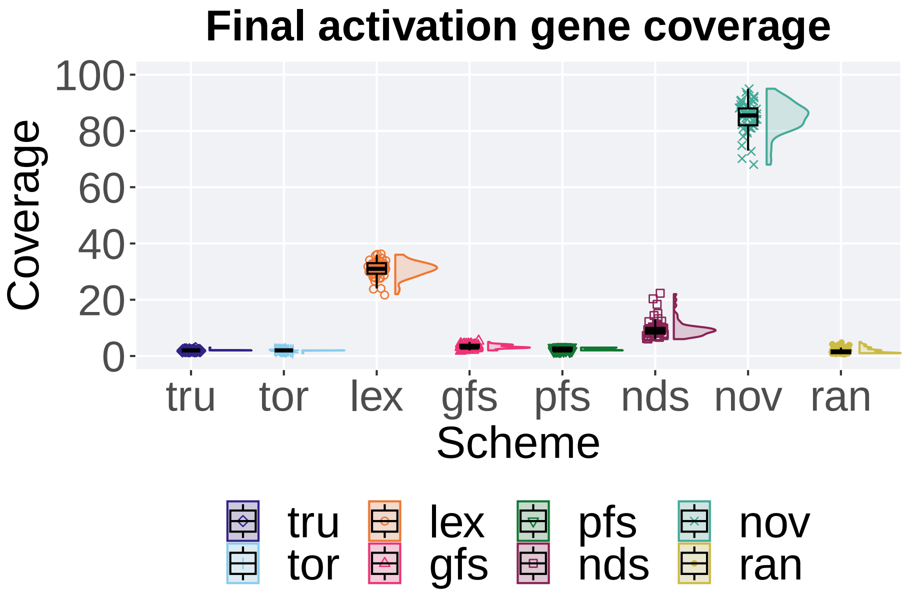
```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Activation gene coverage over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```



5.3.2 End of 50,000 generations

Activation gene coverage in the population at the end of 50,000 generations.

```
# end of run
filter(cc_over_time, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = acron, y = uni_str_pos, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  guides(fill = "none",color = 'none', shape = 'none') +
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Final activation gene coverage')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



5.3.2.1 Stats

Summary statistics for activation gene coverage in the final population.

```
# end of run
coverage = filter(cc_over_time, diagnostic == 'multipath_exploration' & gen == 50000)
coverage$acron = factor(coverage$acron, levels = c('nov','lex','nds','gfs','pfs','tor','tru','ran
coverage %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(uni_str_pos)),
    min = min(uni_str_pos, na.rm = TRUE),
    median = median(uni_str_pos, na.rm = TRUE),
    mean = mean(uni_str_pos, na.rm = TRUE),
    max = max(uni_str_pos, na.rm = TRUE),
    IQR = IQR(uni_str_pos, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
## 1 nov      50      0    68   85.5 84.9     95  6
## 2 lex      50      0    22   31   30.8     36  3.75
## 3 nds      50      0     6    9    9.76    22  2
## 4 gfs      50      0     2    3    3.24     5  1
## 5 pfs      50      0     2    2    2.46     3  1
## 6 tor      50      0     1    2    1.98     2  0
## 7 tru      50      0     2    2    2.02     3  0
## 8 ran      50      0     1    1.5  1.86     5  1
```

Kruskal–Wallis test provides evidence of difference among activation gene coverage in the final population.

```
kruskal.test(uni_str_pos ~ acron, data = coverage)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  uni_str_pos by acron
## Kruskal-Wallis chi-squared = 351.29, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on activation gene coverage in the final population.

```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$acron, p.adjust.method = "bonferroni"
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$uni_str_pos and coverage$acron
```

```
##
##     nov     lex     nds     gfs     pfs     tor     tru
## lex < 2e-16 -       -       -       -       -       -
## nds < 2e-16 < 2e-16 -       -       -       -       -
## gfs < 2e-16 < 2e-16 < 2e-16 -       -       -       -
## pfs < 2e-16 < 2e-16 < 2e-16 7.8e-07 -       -       -
## tor < 2e-16 < 2e-16 < 2e-16 4.2e-16 6.3e-07 -       -
## tru < 2e-16 < 2e-16 < 2e-16 1.4e-15 4.3e-06 1.00000 -
## ran < 2e-16 < 2e-16 < 2e-16 1.1e-08 0.00073 0.20446 0.10598
##
## P value adjustment method: bonferroni
```

5.4 Multi-valley crossing results

5.4.1 Performance

Performance analysis.

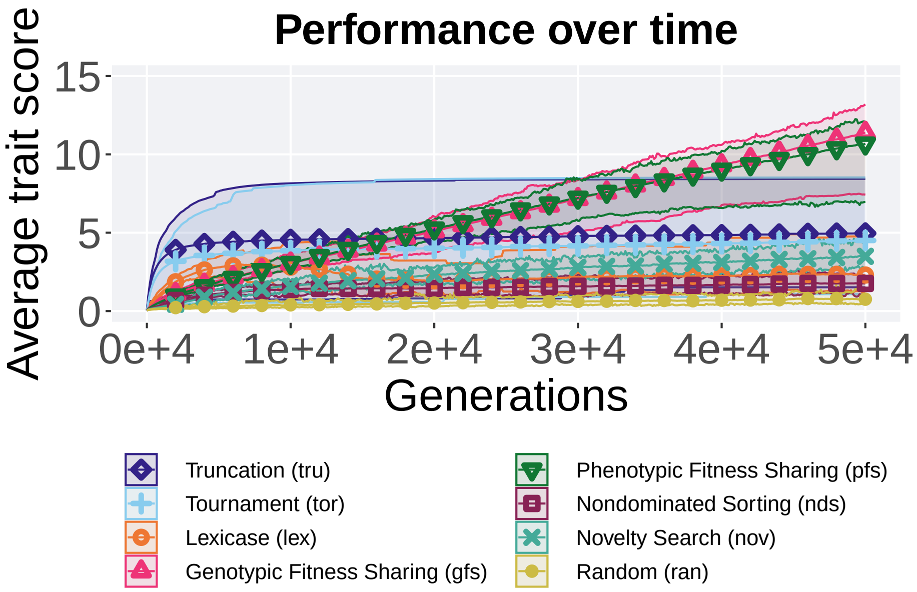
5.4.1.1 Performance over time

Best performance in a population over time.

```
# data for lines and shading on plots
lines = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

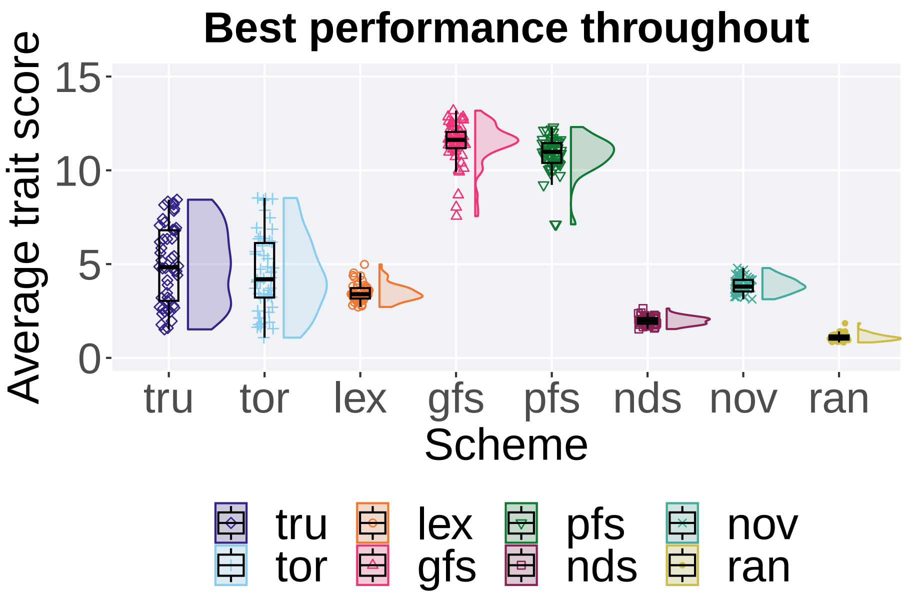
```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 15),
    breaks=seq(0,15, 5),
    labels=c("0", "5", "10", "15")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")

  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Performance over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```



5.4.1.2 Best performance throughout

Best performance found throughout 50,000 generations.

```
### best performance throughout
filter(cc_best_mvc, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape = acron))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  geom_point(position = position_jitter(width = .1), size = 1.5, alpha = 1.0) +
  geom_boxplot(color = 'black', width = .2, outlier.shape = NA, alpha = 0.0) +
  guides(fill = "none",color = 'none', shape = 'none') +
  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 15),
    breaks=seq(0,15, 5),
    labels=c("0", "5", "10", "15")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette, ) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Best performance throughout')+
  p_theme + theme(legend.title=element_blank()) +
  guides(
    shape=guide_legend(nrow=2, title.position = "bottom"),
    color=guide_legend(nrow=2, title.position = "bottom"),
    fill=guide_legend(nrow=2, title.position = "bottom")
  )
```



5.4.1.2.1 Stats

Summary statistics for the performance of the best performance.

```
### best performance throughout
performance = filter(cc_best_mvc, col == 'pop_fit_max' & diagnostic == 'multipath_exploration')
performance$acron = factor(performance$acron, levels = c('gfs','pfs','tor','tru','nov','lex','nds
performance %>%
  group_by(acron) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val / DIMENSIONALITY, na.rm = TRUE),
    median = median(val / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(val / DIMENSIONALITY, na.rm = TRUE),
    max = max(val / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(val / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 8
##   acron count na_cnt   min median  mean   max   IQR
##   <fct> <int>  <int> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 gfs      50      0 7.56    11.6   11.5  13.2  0.865
## 2 pfs      50      0 7.12    11.0   10.8  12.3  1.06
## 3 tor      50      0 1.08     4.19  4.50  8.53 2.91
## 4 tru      50      0 1.52     4.83  4.96  8.43 3.76
## 5 nov      50      0 3.13     3.80  3.88  4.79 0.617
## 6 lex      50      0 2.71     3.39  3.48  4.98 0.555
## 7 nds      50      0 1.54     1.99  1.98  2.63 0.307
## 8 ran      50      0 0.825    1.07  1.10  1.85 0.222
```

Kruskal–Wallis test provides evidence of statistical differences.

```
kruskal.test(val ~ acron, data = performance)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val by acron
## Kruskal-Wallis chi-squared = 335.6, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction.

```
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bonferroni",
                     paired = FALSE, conf.int = FALSE, alternative = 'l')
```

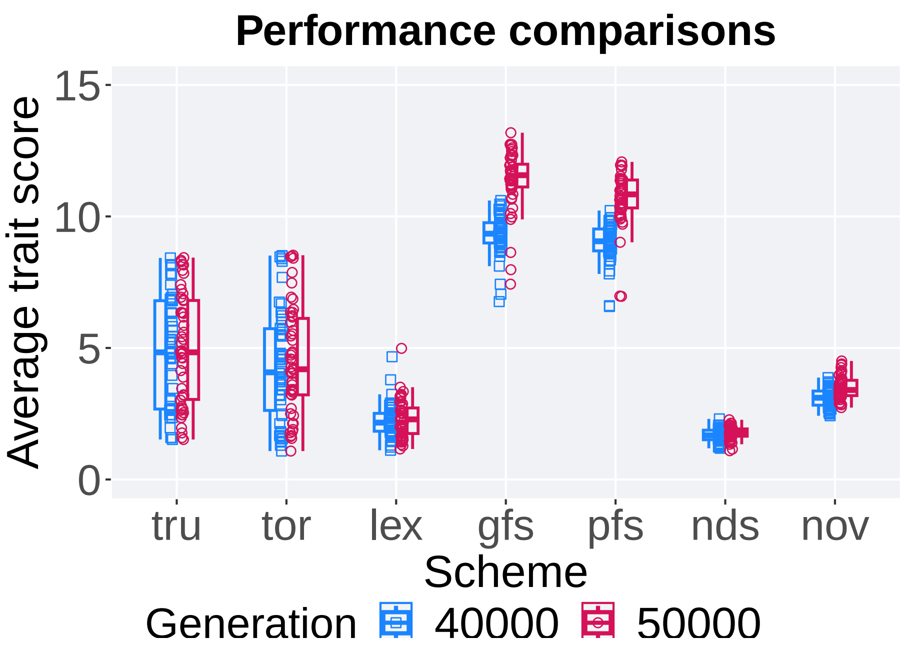
```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  performance$val and performance$acron
##
##     gfs     pfs     tor     tru     nov     lex     nds
## pfs 0.00212 -       -       -       -       -       -
## tor < 2e-16 < 2e-16 -       -       -       -       -
## tru < 2e-16 2.9e-16 1.00000 -       -       -       -
## nov < 2e-16 < 2e-16 1.00000 0.18480 -       -       -
## lex < 2e-16 < 2e-16 0.08240 0.02364 0.00014 -       -
## nds < 2e-16 < 2e-16 2.5e-09 1.7e-12 < 2e-16 < 2e-16 -
## ran < 2e-16 < 2e-16 5.2e-16 < 2e-16 < 2e-16 < 2e-16 2.6e-16
##
## P value adjustment method: bonferroni
```

5.4.1.3 Performance comparison

Best performances in the population at 40,000 and 50,000 generations.

```
## Warning: The following aesthetics were dropped during statistical transformation:
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
## The following aesthetics were dropped during statistical transformation:
## colour, shape
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```

```
# 80% and final generation comparison
end = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & gen == 50000 & a
end$Generation <- factor(end$gen)

mid = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & gen == 40000 & a
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = acron, y=pop_fit_max / DIMENSIONALITY, group = acron, shape
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nud
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1]

  geom_point(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), col = mvc_col
  geom_boxplot(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), position = p

  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 15),
    breaks=seq(0,15, 5),
    labels=c("0", "5", "10", "15")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=c(0,1))+
  scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
  p_theme

plot_grid(
  mvc_p +
    ggtitle("Performance comparisons") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)
```



5.4.1.3.1 Stats

Summary statistics for the performance of the best performance at 40,000 and 50,000 generations.

```
### performance comparisons and generation slices 40K & 50K
slices = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen ==
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
```

```
slices$acron = factor(slices$acron, levels = c('gfs','pfs','tru','tor','nov', 'nds','le
slices %>%
  group_by(acron, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max  / DIMENSIONALITY)),
    min = min(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    median = median(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    mean = mean(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    max = max(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE),
    IQR = IQR(pop_fit_max  / DIMENSIONALITY, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'acron'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 9
## # Groups:   acron [7]
##    acron Generation count na_cnt   min median  mean   max   IQR
##    <fct> <fct>      <int>  <int> <dbl>  <dbl> <dbl> <dbl> <dbl>
##  1 gfs   50000         50      0  7.43  11.6   11.4  13.2  0.865
##  2 gfs   40000         50      0  6.76   9.34  9.32 10.6  0.772
##  3 pfs   50000         50      0  6.96  10.8   10.7  12.1  1.06
##  4 pfs   40000         50      0  6.57   9.06  9.01 10.2  0.832
##  5 tru   50000         50      0  1.52   4.83  4.96  8.43 3.76
##  6 tru   40000         50      0  1.52   4.83  4.85  8.42 4.12
##  7 tor   50000         50      0  1.08   4.19  4.50  8.53 2.91
##  8 tor   40000         50      0  1.08   4.07  4.31  8.51 3.11
##  9 nov   50000         50      0  2.73   3.40  3.49  4.51 0.582
## 10 nov   40000         50      0  2.42   3.11  3.10  3.87 0.538
## 11 nds   50000         50      0  1.09   1.78  1.76  2.27 0.279
## 12 nds   40000         50      0  1.19   1.68  1.68  2.30 0.363
## 13 lex   50000         50      0  1.16   2.29  2.30  4.98 0.974
## 14 lex   40000         50      0  1.11   2.16  2.25  4.67 0.681
```

Truncation selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tru' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'tru' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
## 
##  Wilcoxon rank sum test with continuity correction
## 
## data:  filter(slices, acron == "tru" & Generation == 50000)$pop_fit_max and filter(
## W = 1317, p-value = 0.6466
## alternative hypothesis: true location shift is not equal to 0
```

Tournament selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tor' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'tor' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tor" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 1339, p-value = 0.5418
## alternative hypothesis: true location shift is not equal to 0
```

Lexicase selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'lex' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'lex' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "lex" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 1286, p-value = 0.8067
## alternative hypothesis: true location shift is not equal to 0
```

Genotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'gfs' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'gfs' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "gfs" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 2327, p-value = 1.161e-13
## alternative hypothesis: true location shift is not equal to 0
```

Phenotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'pfs' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'pfs' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "pfs" & Generation == 50000)$pop_fit_max and filter(slices, acr
## W = 2358, p-value = 2.26e-14
## alternative hypothesis: true location shift is not equal to 0
```

Nondominated sorting comparisons.

```
wilcox.test(x = filter(slices, acron == 'nds' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'nds' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nds" & Generation == 50000)$pop_fit_max and filter(
## W = 1509, p-value = 0.07474
## alternative hypothesis: true location shift is not equal to 0
```

Novelty search comparisons.

```
wilcox.test(x = filter(slices, acron == 'nov' & Generation == 50000)$pop_fit_max,
            y = filter(slices, acron == 'nov' & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nov" & Generation == 50000)$pop_fit_max and filter(
## W = 1872, p-value = 1.831e-05
## alternative hypothesis: true location shift is not equal to 0
```

5.4.2 Activation gene coverage

Activation gene coverage analysis.

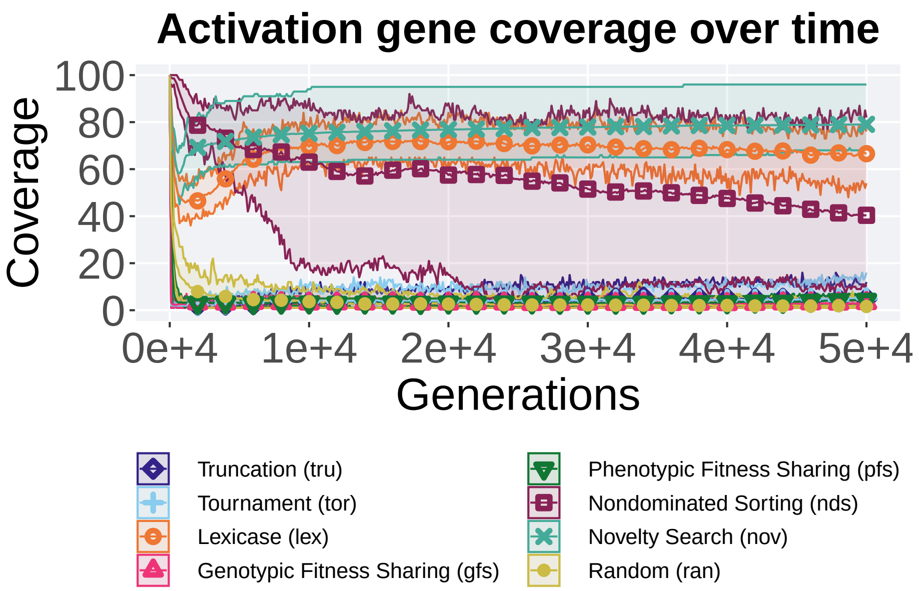
5.4.2.1 Coverage over time

Activation gene coverage over time.

```
# data for lines and shading on plots
lines = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )
```

```
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the `.groups` argument.
```

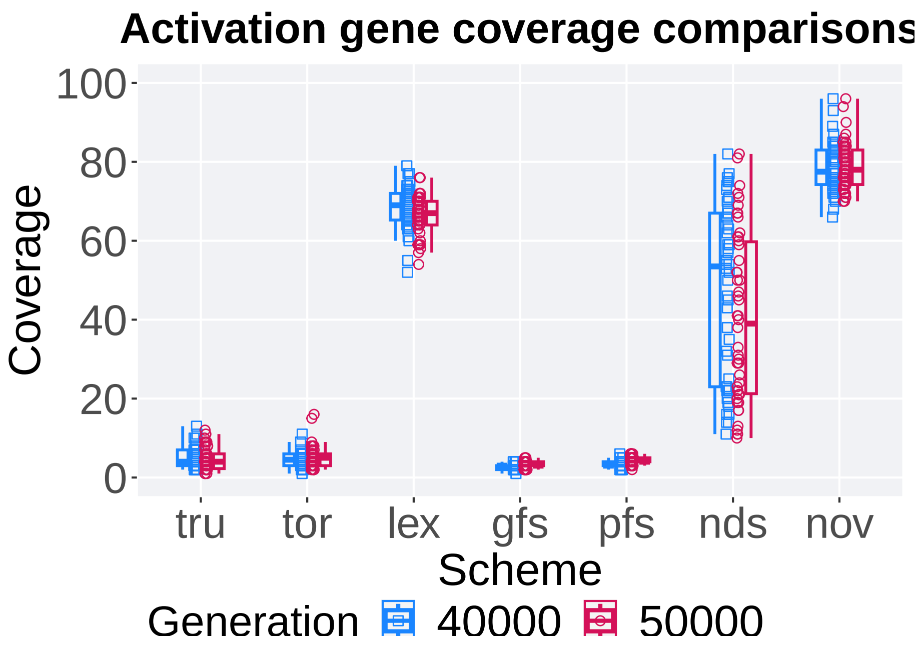
```
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill =`Selection\nScheme
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_continuous(
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +
  scale_shape_manual(values=SHAPE)+
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle('Activation gene coverage over time')+
  p_theme + theme(legend.title=element_blank(),legend.text=element_text(size=11)) +
  guides(
    shape=guide_legend(ncol=2, title.position = "bottom"),
    color=guide_legend(ncol=2, title.position = "bottom"),
    fill=guide_legend(ncol=2, title.position = "bottom")
  )
```



5.4.2.2 Coverage comparison

Best activation gene coverage in the population at 40,000 and 50,000 generations.

```
# 80% and final generation comparison
end = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & gen == 50000 & a
end$Generation <- factor(end$gen)

mid = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & gen == 40000 & a
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = acron, y=uni_str_pos, group = acron, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nuc
  geom_boxplot(position = position_nudge(x = -.17, y = 0), lwd = 0.7, col = mvc_col[1]

  geom_point(data = end, aes(x = acron, y=uni_str_pos), col = mvc_col[2], position = po
  geom_boxplot(data = end, aes(x = acron, y=uni_str_pos), position = position_nudge(x =

  scale_y_continuous(
    name="Coverage",
    limits=c(0, 100),
    breaks=seq(0,100, 20),
    labels=c("0", "20", "40", "60", "80", "100")
  ) +
  scale_x_discrete(
    name="Scheme"
  )+
  scale_shape_manual(values=c(0,1))+
  scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
  p_theme

plot_grid(
  mvc_p +
    ggtitle("Activation gene coverage comparisons") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)
```



5.4.2.3 Stats

Summary statistics for the activation gene coverage at 40,000 and 50,000 generations.

```
slices = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen ==
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices$acron = factor(slices$acron, levels = c('nov','lex', 'nds','tru','tor','gfs','pfs','ran'))
slices %>%
  group_by(acron, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(uni_str_pos)),
    min = min(uni_str_pos, na.rm = TRUE),
    median = median(uni_str_pos, na.rm = TRUE),
    mean = mean(uni_str_pos, na.rm = TRUE),
    max = max(uni_str_pos, na.rm = TRUE),
    IQR = IQR(uni_str_pos, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'acron'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 9
## # Groups:   acron [7]
##    acron Generation count na_cnt   min median  mean   max   IQR
##    <fct> <fct>      <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
##  1 nov   50000         50      0    70   78   79.1     96  8.75
##  2 nov   40000         50      0    66   77.5 78.6     96  8.75
##  3 lex   50000         50      0    54   67   66.6     76  6
##  4 lex   40000         50      0    52   69   68.3     79  6.75
##  5 nds   50000         50      0    10   39   40.4     82 38.5
##  6 nds   40000         50      0    11   53.5 47.5     82 44
##  7 tru   50000         50      0     1    4    4.8     12  3.75
##  8 tru   40000         50      0     2    4    4.78    13  4
##  9 tor   50000         50      0     2    5    5.1     16  3
## 10 tor   40000         50      0     1    4.5  4.38    11  3
## 11 gfs   50000         50      0     2    3    3.14     5  1
## 12 gfs   40000         50      0     1    3    2.72     4  1
## 13 pfs   50000         50      0     2    4    4.34     6  1
## 14 pfs   40000         50      0     2    3    3.28     6  1
```

Truncation selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tru' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'tru' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tru" & Generation == 50000)$uni_str_pos and filter(
## W = 1254.5, p-value = 0.9778
## alternative hypothesis: true location shift is not equal to 0
```

Tournament selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'tor' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'tor' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "tor" & Generation == 50000)$uni_str_pos and filter(
## W = 1396, p-value = 0.3094
## alternative hypothesis: true location shift is not equal to 0
```

Lexicase selection comparisons.

```
wilcox.test(x = filter(slices, acron == 'lex' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'lex' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
```

```
##
## data:  filter(slices, acron == "lex" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 992.5, p-value = 0.07568
## alternative hypothesis: true location shift is not equal to 0
```

Genotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'gfs' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'gfs' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "gfs" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 1573, p-value = 0.01769
## alternative hypothesis: true location shift is not equal to 0
```

Phenotypic fitness sharing comparisons.

```
wilcox.test(x = filter(slices, acron == 'pfs' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'pfs' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "pfs" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 1914.5, p-value = 2.023e-06
## alternative hypothesis: true location shift is not equal to 0
```

Nondominated sorting comparisons.

```
wilcox.test(x = filter(slices, acron == 'nds' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'nds' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nds" & Generation == 50000)$uni_str_pos and filter(slices, acr
## W = 1008, p-value = 0.09584
## alternative hypothesis: true location shift is not equal to 0
```

Novelty search comparisons.

```
wilcox.test(x = filter(slices, acron == 'nov' & Generation == 50000)$uni_str_pos,
            y = filter(slices, acron == 'nov' & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##
```

```
##  Wilcoxon rank sum test with continuity correction
##
## data:  filter(slices, acron == "nov" & Generation == 50000)$uni_str_pos and filter(
## W = 1295.5, p-value = 0.756
## alternative hypothesis: true location shift is not equal to 0
```