

Diagnostics Supplemental Material

Jose Guadalupe Hernandez

2023-01-31

Contents

1	Introduction	5
1.1	About our supplemental material	5
1.2	Contributing authors	6
1.3	Research overview	6
1.4	Computer Setup	6
1.5	Experimental setup	7
2	Exploitation rate results	23
2.1	Analysis dependencies	23
2.2	Performance over time	23
2.3	Best performance throughout	25
2.4	Generation satisfactory solution found	28
2.5	Multi-valley crossing results	30
3	Ordered exploitation results	41
3.1	Analysis dependencies	41
3.2	Performance over time	41
3.3	Best performance throughout	43
3.4	Generation satisfactory solution found	45
3.5	Multi-valley crossing results	48
4	Contradictory objectives results	59
4.1	Analysis dependencies	59
4.2	Satisfactory trait coverage	59
4.3	Activation gene coverage	66
4.4	Nondominated sorting split	71
4.5	Multi-valley crossing results	76
5	Multi-path mpeloration results	91
5.1	Analysis dependencies	91
5.2	Performance	91
5.3	Activation gene coverage	98
5.4	Multi-valley crossing results	102

6 Truncation selection	117
6.1 Exploitation rate results	117
6.2 Ordered exploitation results	126
6.3 Contradictory objectives diagnostic	136
6.4 Multi-path exploration results	156
7 Tournament selection	177
7.1 Exploitation rate results	177
7.2 Ordered exploitation results	186
7.3 Contradictory objectives diagnostic	196
7.4 Multi-path exploration results	215
8 Genotypic fitness sharing	237
8.1 Exploitation rate results	237
8.2 Ordered exploitation results	247
8.3 Contradictory objectives diagnostic	254
8.4 Multi-path exploration results	273
9 Phenotypic fitness sharing	295
9.1 Exploitation rate results	295
9.2 Ordered exploitation results	303
9.3 Contradictory objectives diagnostic	311
9.4 Multi-path exploration results	331
10 Nondominated sorting	353
10.1 Exploitation rate results	353
10.2 Ordered exploitation results	361
10.3 Contradictory objectives diagnostic	369
10.4 Multi-path exploration results	389
11 Novelty Search	411
11.1 Exploitation rate results	411
11.2 Ordered exploitation results	420
11.3 Contradictory objectives diagnostic	428
11.4 Multi-path exploration results	447

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 6.4)
- Multi-valley crossing results (Section 2.5)

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

- Truncation selection (Section 6)
- Tournament selection sharing (Section 7)
- Genotypic fitness sharing (Section 8)
- Phenotypic fitness sharing (Section 9)
- Nondominated sorting (Section 10)
- Novelty search (Section 11)

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 tidyverse 1.3.0     vforcats 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 Selection (gfs)', 'Parity Filter (pfs)', 'Nearest Descent (nnd)', 'Novelty (nov)', 'Random (ran)')
SCHEME = c('TRUNCATION', 'TOURNAMENT', 'LEXICASE', 'FITSHARING_G', 'FITSHARING_P', 'NONDOMINANT_SELECTION')
ORDER = c('Truncation (tru)', 'Tournament (tor)', 'Lexicase (lex)', 'Genotypic Fitness Selection (gfs)', 'Parity Filter (pfs)', 'Nearest Descent (nnd)', 'Novelty (nov)', 'Random (ran)')

```

```

# 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', 'NO-MVC', 'MVC', 'NO-MVC-MVC', 'MVC-NO-MVC', 'MVC-MVC'))

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

## [1] "Collecting over time data..."
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]), '.csv')
    dir_mvc = paste(DATA_DIR, 'MVC/', SCHEME[i], '/over-time-', 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_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$arc_acti_gene - cc_over_time$cc
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$arc_acti_gene - cc_over_time_mvc$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',sep='')

    # 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(SCHEM
    dir_mvc = paste(DATA_DIR, 'MVC/', SCHEME[i], '/over-time-', diagnostic, '-', tolower(SCHEM
  }
}

```

```

# 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$arc_acti_gene - ss_over_time$arc_acti_gene
ss_over_time = subset(ss_over_time, select = -c(pop_fit_avg,archive_cnt,pmin,pareto_cnt,arc_acti_gene))

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

## 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', sep = '')
# 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
tor_ssf = rbind(exp_df,ord_df)
tor_ssf$T = tor_ssf$trt
tor_ssf$T = factor(tor_ssf$T)

#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', sep = '')
# 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)
tru_ssf$T = tru_ssf$trt
tru_ssf$T = factor(tru_ssf$T)

```

```
#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

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

2.2 Performance over time

Best performance in a population over time.

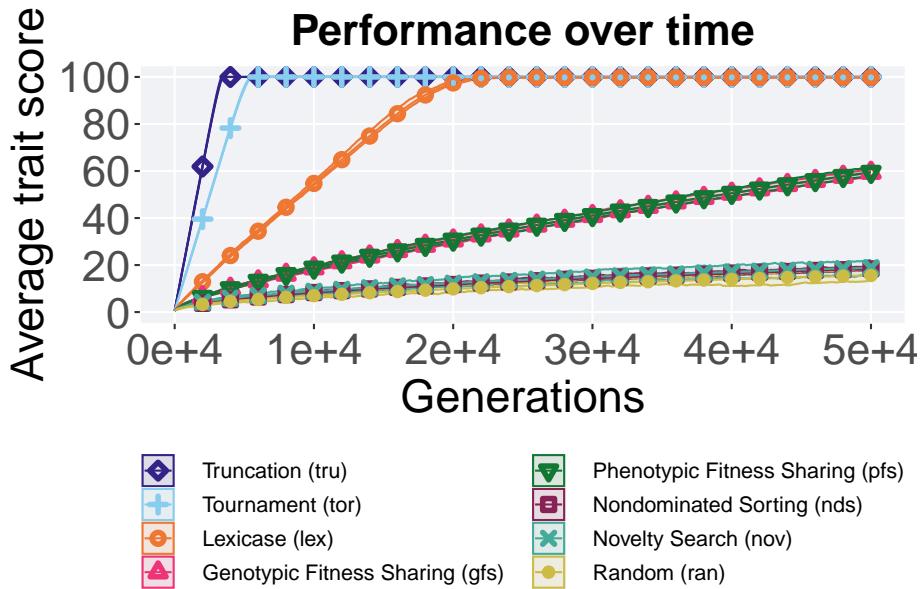
```
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'exploitation_rate') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY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

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

```

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.

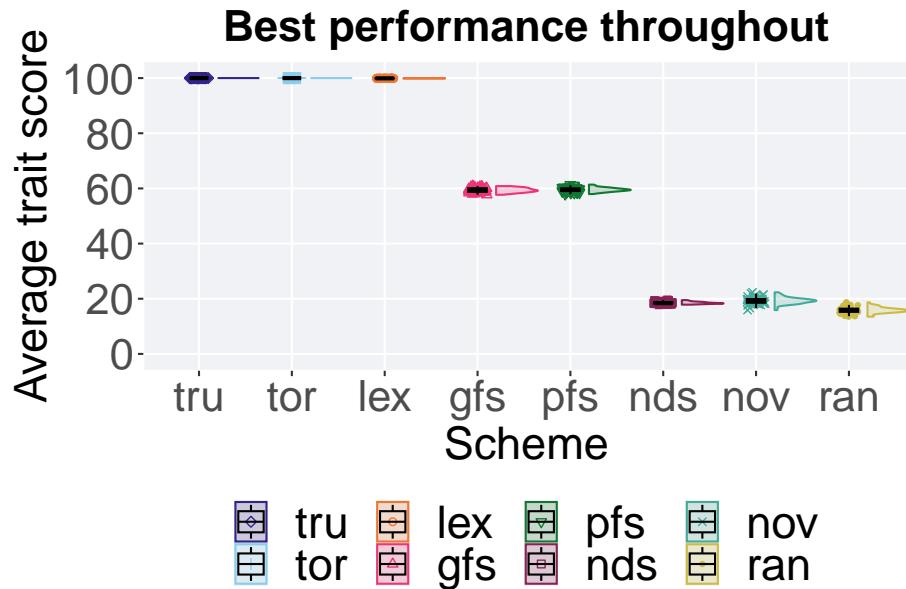
```
### best performance throughout
filter(cc_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape = acron))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  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(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` aesthetic with geom_polygon was deprecated in ggplot2 3.4.0
## 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', 'pfs', 'nds', 'nov', 'ran'))
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.

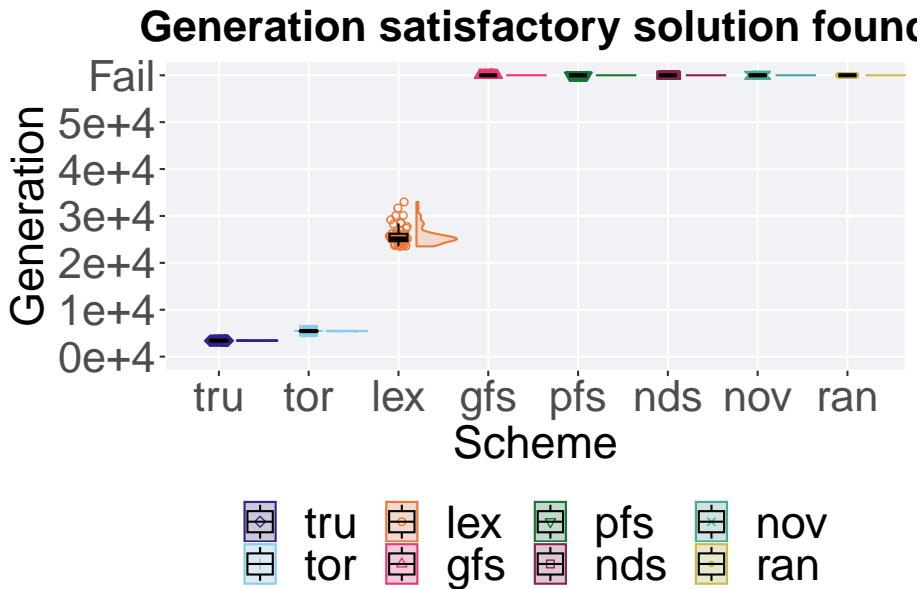
```
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = '1')
```

```
##
## 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 = 0.5) +
  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> <dbl>  <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,
             name="Average trait score",

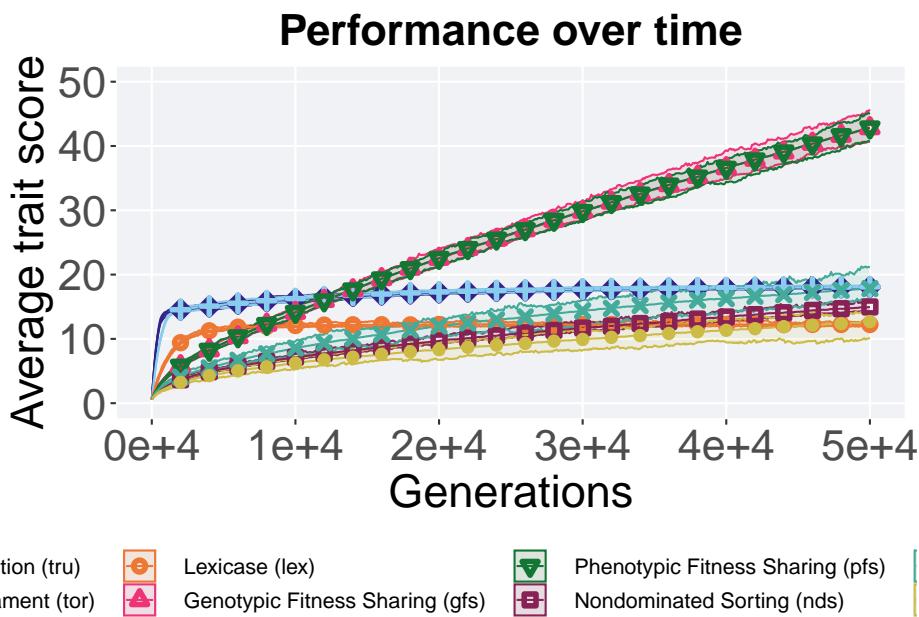
```

```

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"),
  fill=guide_legend(ncol=2, title.position = "left")
)

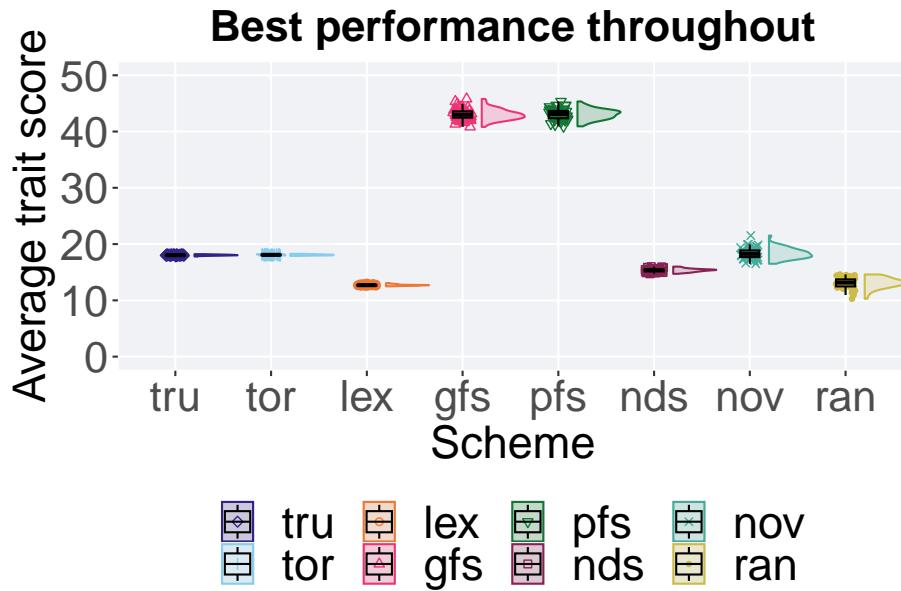
```



2.5.2 Best performance throughout

Best performance found throughout 50,000 generations.

```
### 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','lex'))
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 = "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     tru     tor     nov     nds     lex
## pfs 1      -      -      -      -      -      -
## tru <2e-16 <2e-16 1      -      -      -      -
## 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.

```
dummy_df = filter(filter(cc_over_time_mvc, diagnostic == 'exploitation_rate' & (gen == 40000 | gen == 50000)), .by = c('acron', 'gen'))
dummy_df$Generation <- factor(dummy_df$gen, levels = c(50000, 40000))
p = ggplot(dummy_df, aes(x=gen, y=pop_fit_max / DIMENSIONALITY, group = acron, fill = acron))
  p + geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
    geom_point(position = position_jitter(width = .03), size = 2, alpha = 1.0) +
    geom_boxplot(position = position_nudge(x = .13, y = 0), lwd = 1, width = .1, outlier = FALSE) +
    scale_shape_manual(values=c(0,1)) +
    scale_colour_manual(values = mvc_col) +
    scale_fill_manual(values = cb_palette) + p_theme +
```

```

guides(
  shape=guide_legend(ncol=2, title.position = "bottom"),
  color=guide_legend(ncol=2, title.position = "bottom"),
  fill='none')

legend <- cowplot::get_legend(
  p +
  guides(
    shape=guide_legend(ncol=2, title.position = "left"),
    color=guide_legend(ncol=2, title.position = "left"),
    fill='none'
  ) +
  theme(
    legend.position = "top",
    legend.box="vertical",
    legend.justification="center"
  )
)

## 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?

rm(dummy_df, p)

# 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 = Generation))
  geom_point(col = mvc_col[1], position = position_jitternudge(jitter.width = .03, nudge = 0))
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = 'white')
  geom_point(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2])

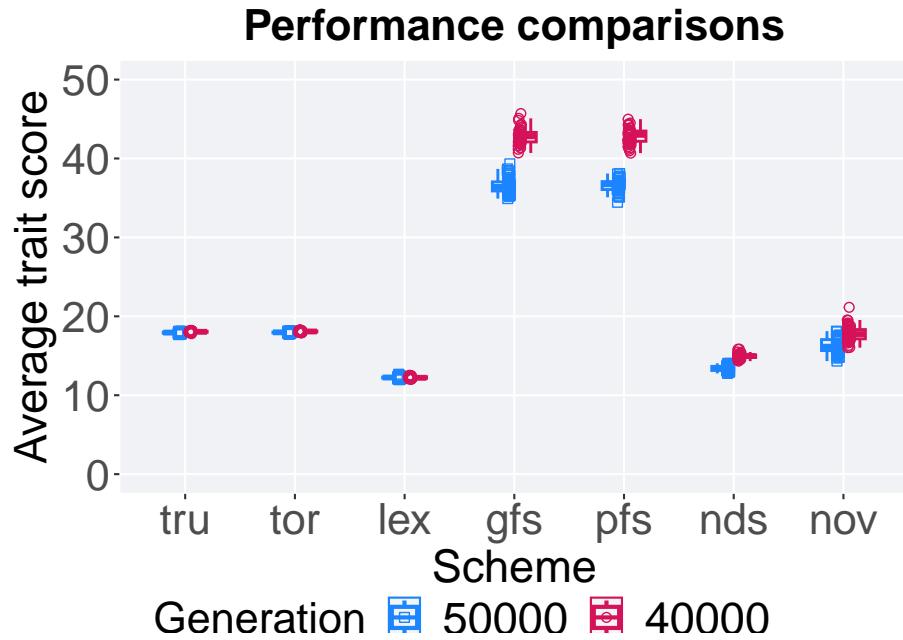
```

```

geom_boxplot(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), pos=1) +
  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 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices$acron = factor(slices$acron, levels = c('gfs','pfs','tru','tor','nov', 'nds','lex', 'ran'))
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>
## 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(  

## 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(  

## 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(  

## 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(  

## 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(slices, acr
## 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(slices, acr
## 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

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

3.2 Performance over time

Best performance in a population over time.

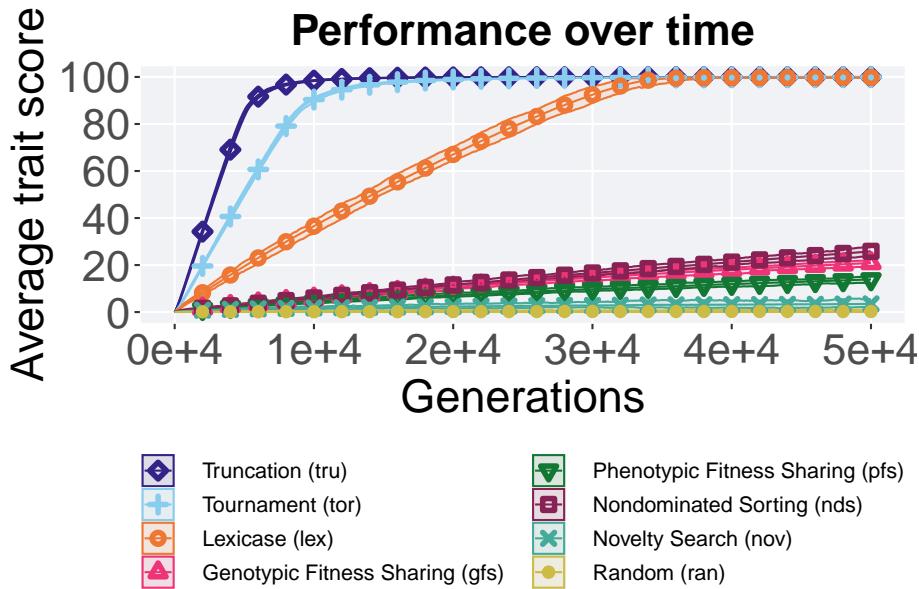
```
# data for lines and shading on plots
lines = filter(cc_over_time, diagnostic == 'ordered_exploitation') %>%
  group_by(`Selection\nScheme`, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY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

## `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")
  )

```

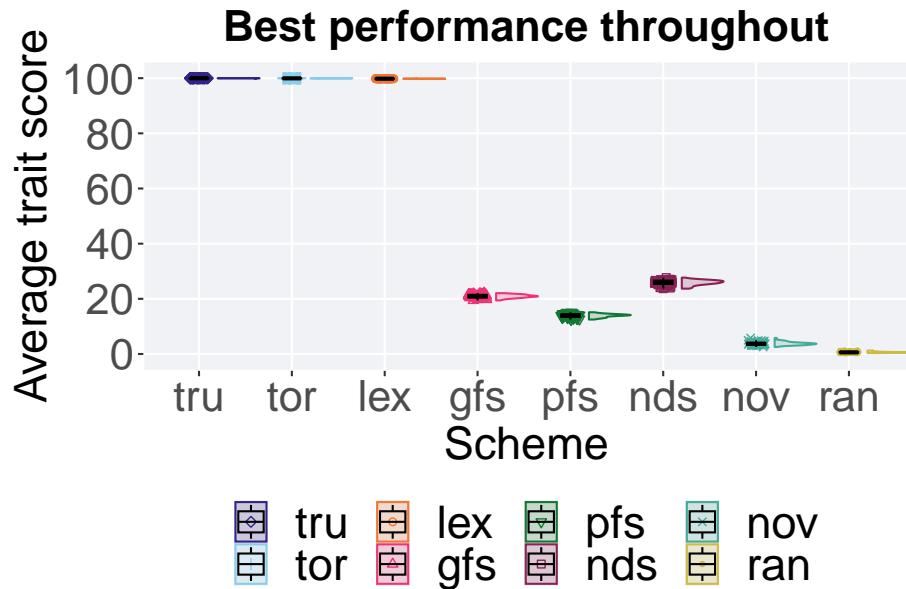


3.3 Best performance throughout

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_exploitation')
performance$acron = factor(performance$acron, levels = c('tru', 'tor', 'lex', '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 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.

```
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.

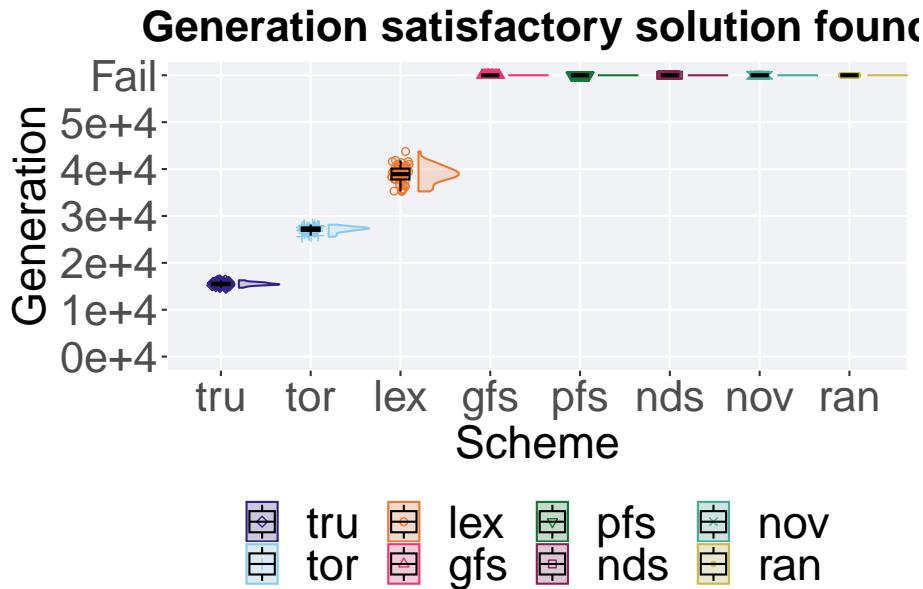
```
pairwise.wilcox.test(x = performance$val, g = performance$acron, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 'l')

##
## 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 = 0.5)
  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> <dbl> <dbl> <dbl> <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.

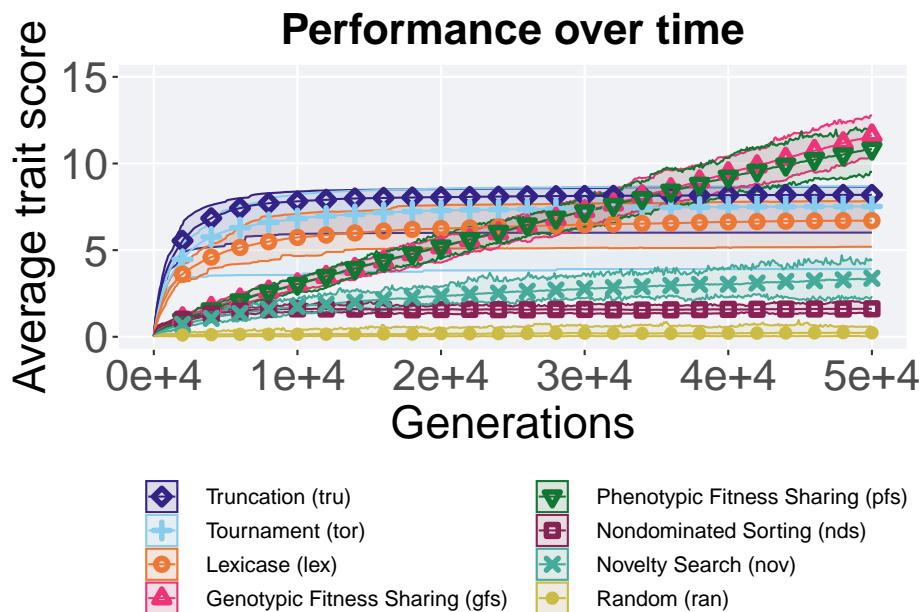
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill = `Selection\nScheme`,
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2,
  scale_y_continuous(
```

```

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")
)

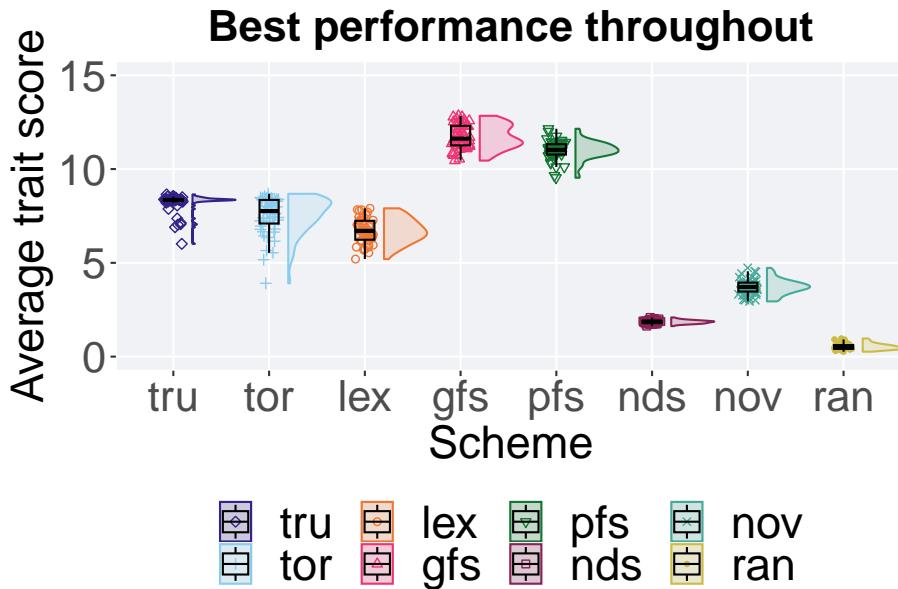
```



3.5.2 Best performance throughout

Best performance found throughout 50,000 generations.

```
### 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 =
      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 = "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      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)

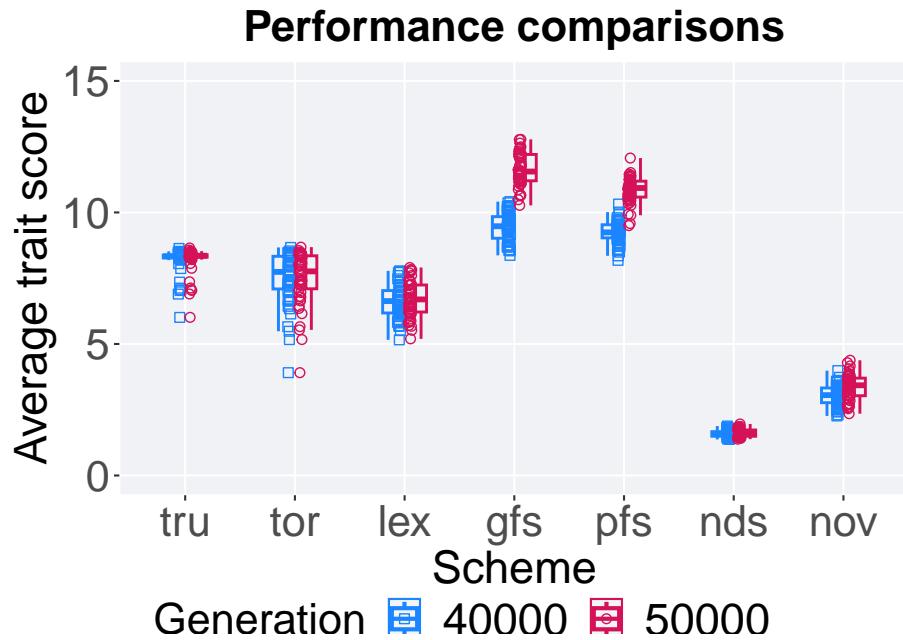
mid = filter(cc_over_time_mvc, diagnostic == 'ordered_exploitation' & gen == 40000 & acron != 'ra')
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = acron, y=pop_fit_max / DIMENSIONALITY, group = acron, shape = Generation))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 10)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])
  geom_point(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = -.15, y = 0))
  geom_boxplot(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 0))

  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 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000, 40000))
slices$acron = factor(slices$acron, levels = c('gfs', 'pfs', 'tru', 'tor', 'lex', 'nov', 'ncl', 'ndc', 'ndt', 'ndv', 'ndw'))
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
##   acron Generation count na_cnt min median mean max IQR
##   <fct>   <dbl>   <int>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 gfs      50000     14     13  12.5  11.5  12.0  12.5  13.0
## 2 gfs      40000     14     13  12.5  11.5  12.0  12.5  13.0
## 3 pfs      50000     14     13  12.5  11.5  12.0  12.5  13.0
## 4 pfs      40000     14     13  12.5  11.5  12.0  12.5  13.0
## 5 tru      50000     14     13  12.5  11.5  12.0  12.5  13.0
## 6 tru      40000     14     13  12.5  11.5  12.0  12.5  13.0
## 7 tor      50000     14     13  12.5  11.5  12.0  12.5  13.0
## 8 tor      40000     14     13  12.5  11.5  12.0  12.5  13.0
## 9 lex      50000     14     13  12.5  11.5  12.0  12.5  13.0
## 10 lex     40000     14     13  12.5  11.5  12.0  12.5  13.0
## 11 nov     50000     14     13  12.5  11.5  12.0  12.5  13.0
## 12 nov     40000     14     13  12.5  11.5  12.0  12.5  13.0
## 13 ncl     50000     14     13  12.5  11.5  12.0  12.5  13.0
## 14 ncl     40000     14     13  12.5  11.5  12.0  12.5  13.0
## 15 ndc     50000     14     13  12.5  11.5  12.0  12.5  13.0
## 16 ndc     40000     14     13  12.5  11.5  12.0  12.5  13.0
## 17 ndt     50000     14     13  12.5  11.5  12.0  12.5  13.0
## 18 ndt     40000     14     13  12.5  11.5  12.0  12.5  13.0
## 19 ndv     50000     14     13  12.5  11.5  12.0  12.5  13.0
## 20 ndv     40000     14     13  12.5  11.5  12.0  12.5  13.0
## 21 ndw     50000     14     13  12.5  11.5  12.0  12.5  13.0
## 22 ndw     40000     14     13  12.5  11.5  12.0  12.5  13.0
```

```

##   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.

```
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.

```
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.

```
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(
## 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(
## 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(
## 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(
## 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 coverage** 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(PupilometryR)
library(sdamr)
```

4.2 Satisfactory trait coverage

Satisfactory trait coverage analysis.

4.2.1 Coverage over time

Satisfactory trait coverage over time.

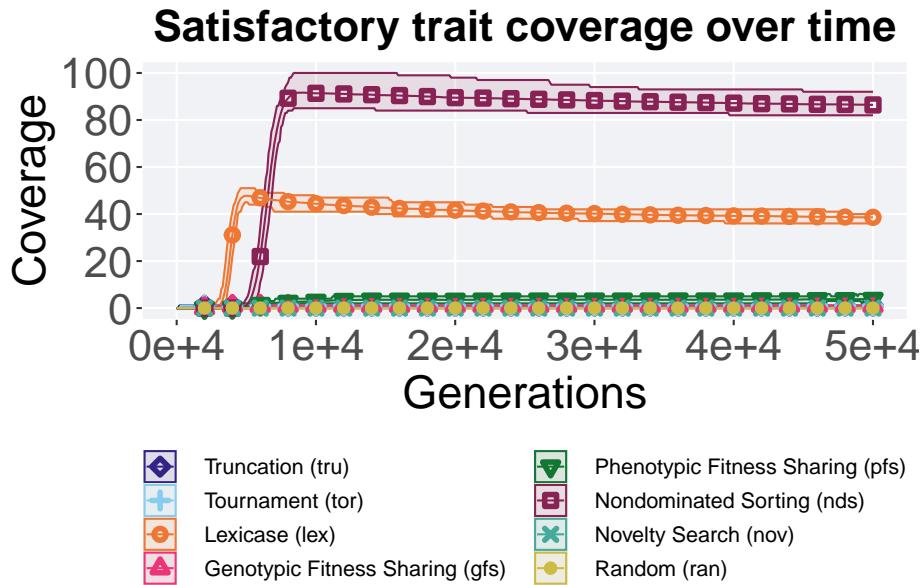
```

# 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.

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.

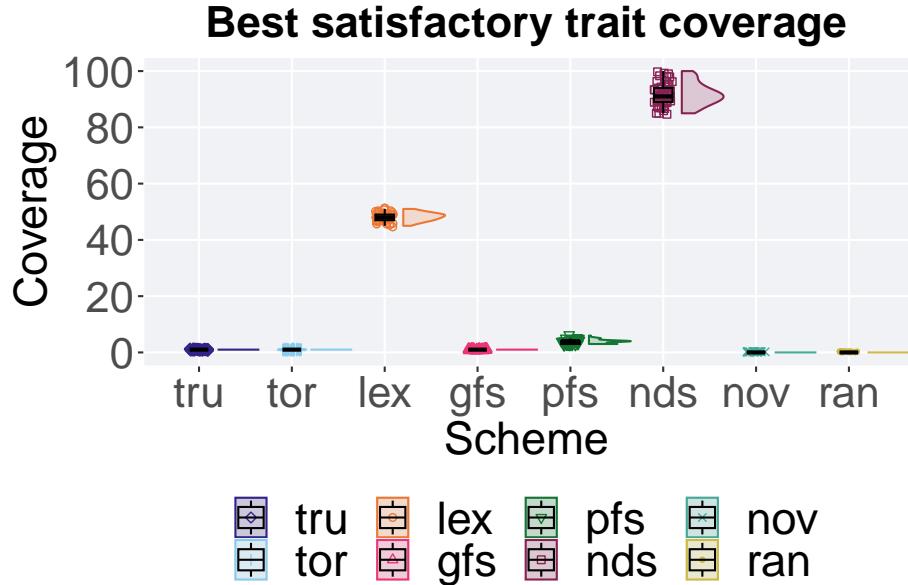
```
### best satisfactory trait coverage throughout
filter(cc_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = acron, y = val, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 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 60 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_objective')
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>
## 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.

```
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.

```
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

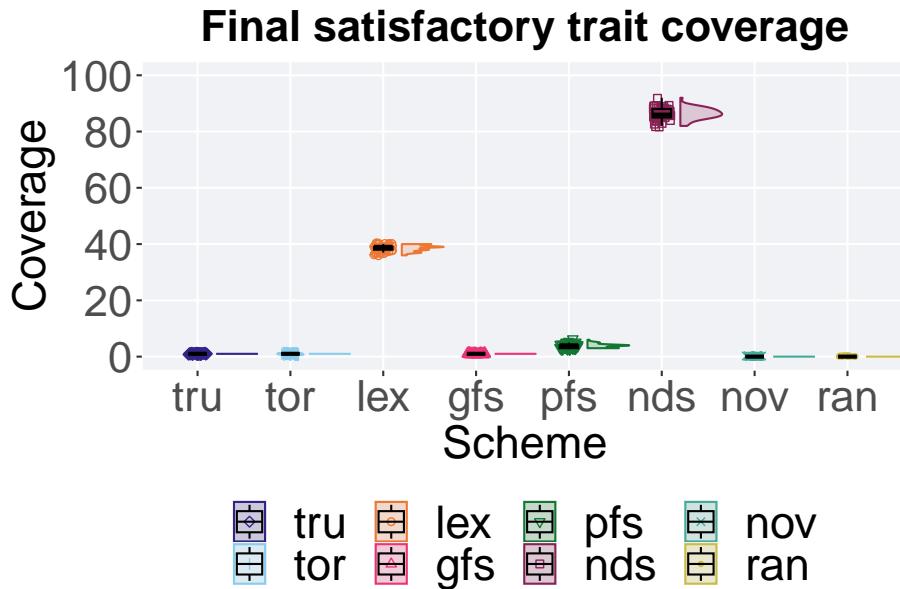
Satisfactory trait coverage in the population at the end of 50,000 generations.

```

#### 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 = 0.5) +
  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 56 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> <dbl> <dbl> <dbl> <dbl>
## 1 nds     50      0    82    86.4  86.4    92     3
## 2 lex     50      0    36    39.0  38.6    40     1
## 3 pfs     50      0     3     4.0  3.82     6     1
```

```
## 4 gfs      50     0     1     1     1     0
## 5 tor      50     0     1     1     1     0
## 6 tru      50     0     1     1     1     0
## 7 nov      50     0     0     0     0     0
## 8 ran      50     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 = "bonferroni",
                      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.

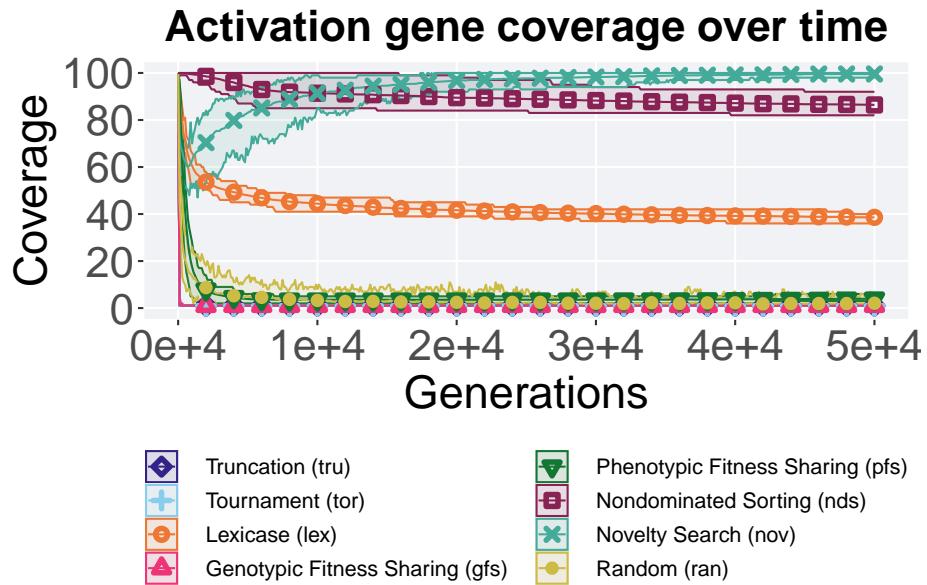
```

# 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.

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

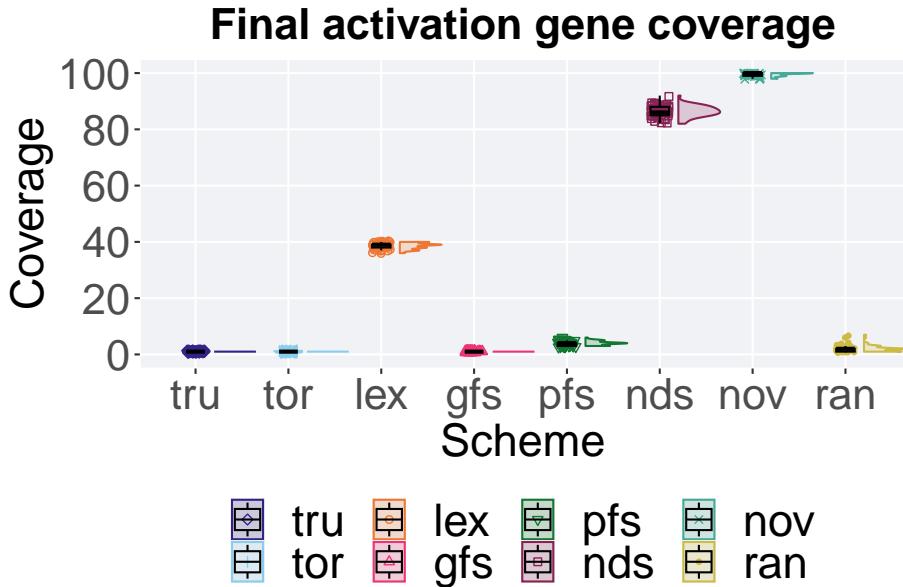
Activation gene coverage in the population at the end of 50,000 generations.

```
# 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 = 0.2),
    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('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 14 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 = "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    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.

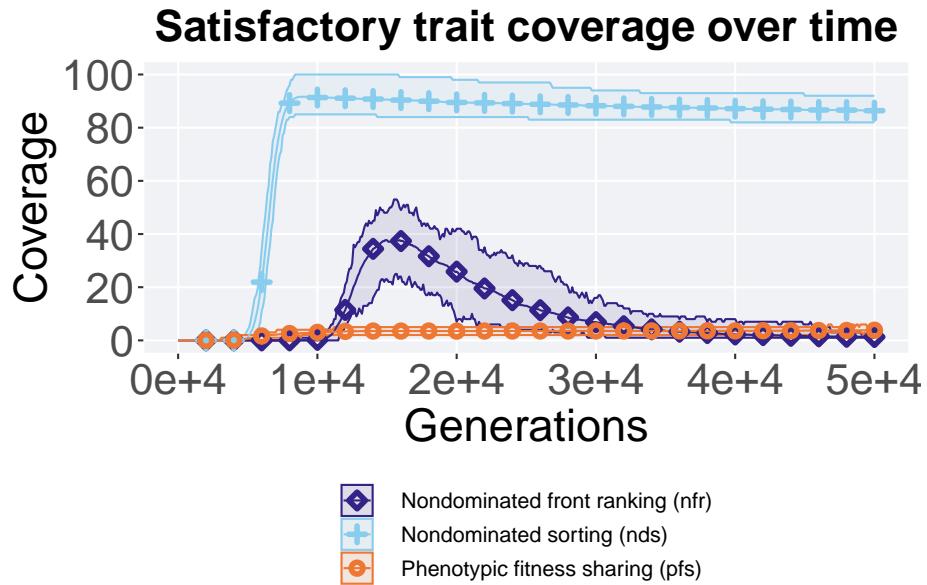
4.4.1 Coverage over time

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`, 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

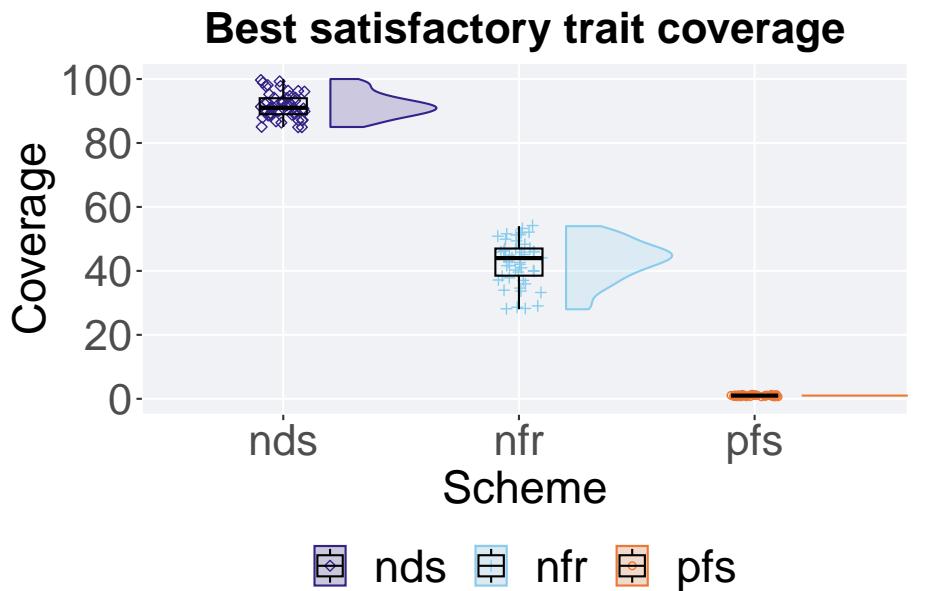
Best satisfactory trait coverage.

```
### 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 = 0.5) +
  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 = "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     pfs
## pfs <2e-16 -
## nfr <2e-16 1
##
## P value adjustment method: bonferroni
```

4.4.3 End of 50,000 generations

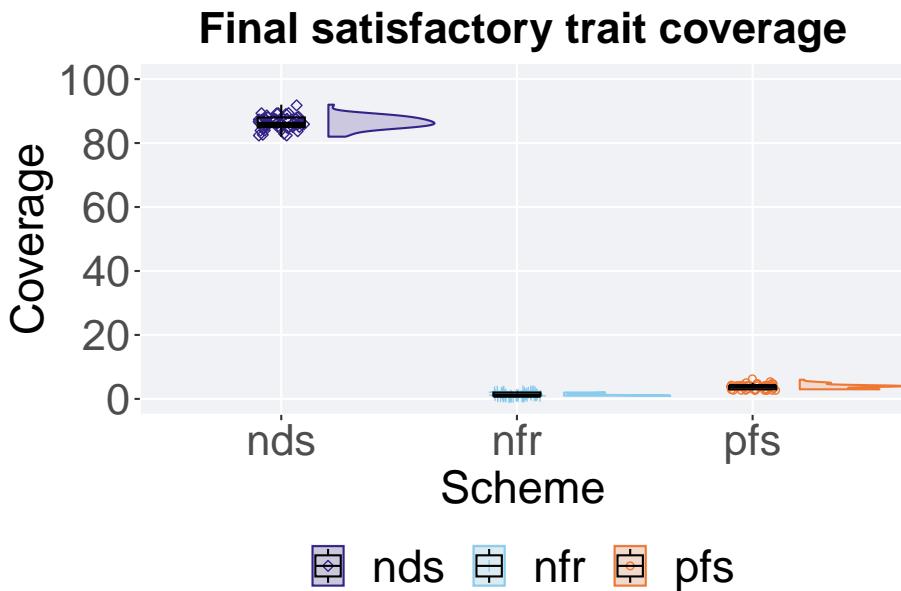
Satisfactory trait coverage in the population at the end of 50,000 generations.

```
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 = 0.5) +
  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),
    max = max(pop_uni_obj),
    mean = mean(pop_uni_obj),
    median = median(pop_uni_obj),
    sd = sd(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> <dbl> <dbl> <dbl> <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 = "Tukey"
                      paired = FALSE, conf.int = FALSE, alternative = '1')

##
## 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.

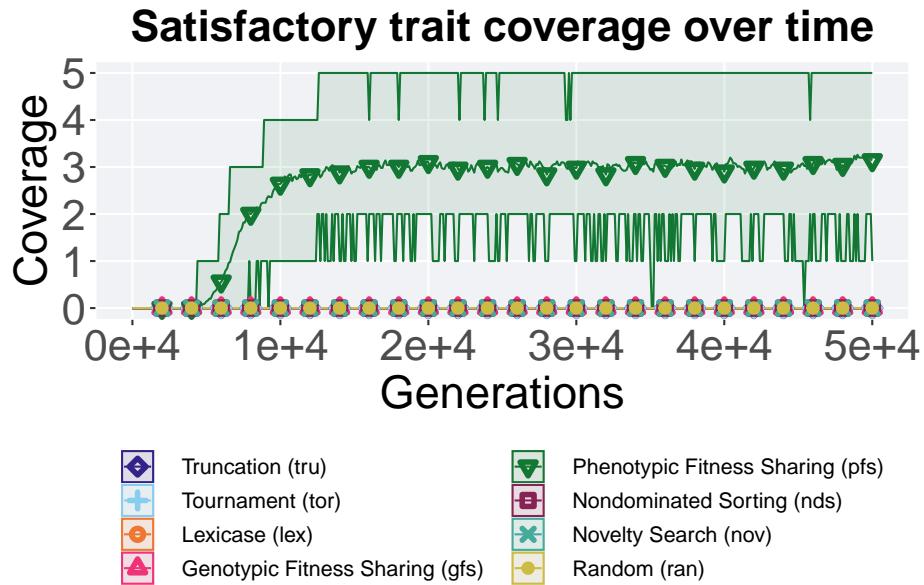
4.5.1.1 Coverage over time

Satisfactory trait coverage over time.

```
# 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.

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.

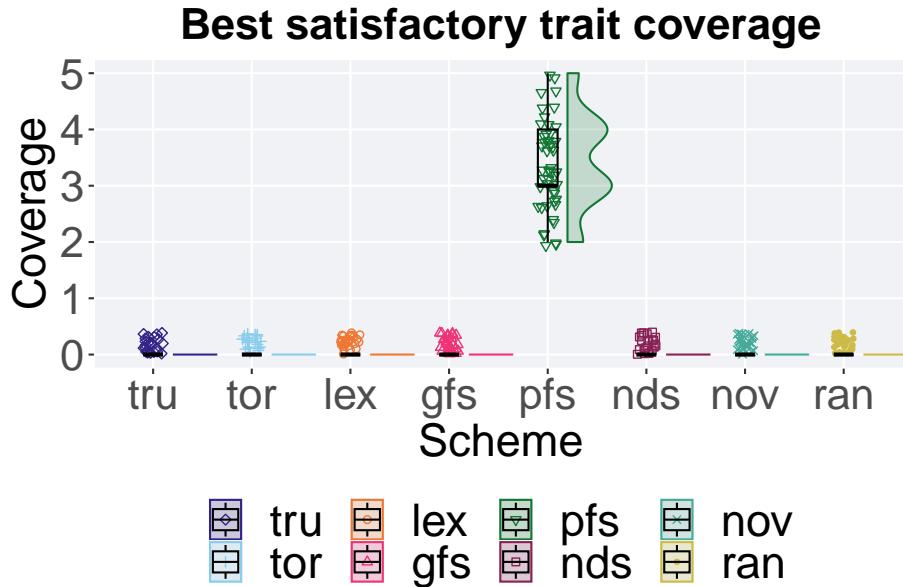
```
### best satisfactory trait coverage throughout
filter(cc_best_mvc, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = acron, y = val, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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_colorbar()
  )
```

```

color=guide_legend(nrow=2, title.position = "bottom"),
fill=guide_legend(nrow=2, title.position = "bottom")
)

## Warning: Removed 171 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     0
## 3 lex    50     0     0     0     0     0     0     0
## 4 gfs    50     0     0     0     0     0     0     0
## 5 tor    50     0     0     0     0     0     0     0
## 6 tru    50     0     0     0     0     0     0     0
## 7 nov    50     0     0     0     0     0     0     0
## 8 ran    50     0     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 = "bonferroni"
                      paired = FALSE, conf.int = FALSE, alternative = '1')

##
## 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 != 'A')
end$Generation <- factor(end$gen)

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

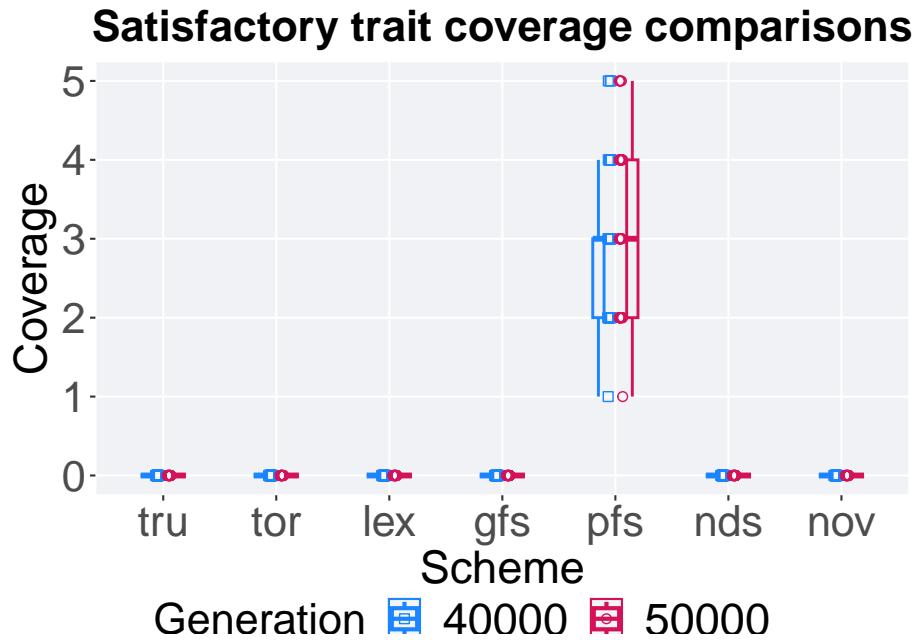
mvc_p = ggplot(mid, aes(x = acron, y=pop_uni_obj, group = acron, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), nudge.y = 0) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = acron, y=pop_uni_obj), col = mvc_col[2], position = position_jitternudge(jitter.width = .03, nudge.x = 0.15), nudge.y = 0) +
  geom_boxplot(data = end, aes(x = acron, y=pop_uni_obj), position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

  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 == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000, 40000))
slices$acron = factor(slices$acron, levels = c('pfs', 'nd', '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
## `.` argument.

## # A tibble: 14 x 9
## # Groups:   acron [7]
##   acron Generation count na_cnt   min median   mean   max     IQR
#>   <fct>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

##   <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.  

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.  

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.  

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.  

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)
  )

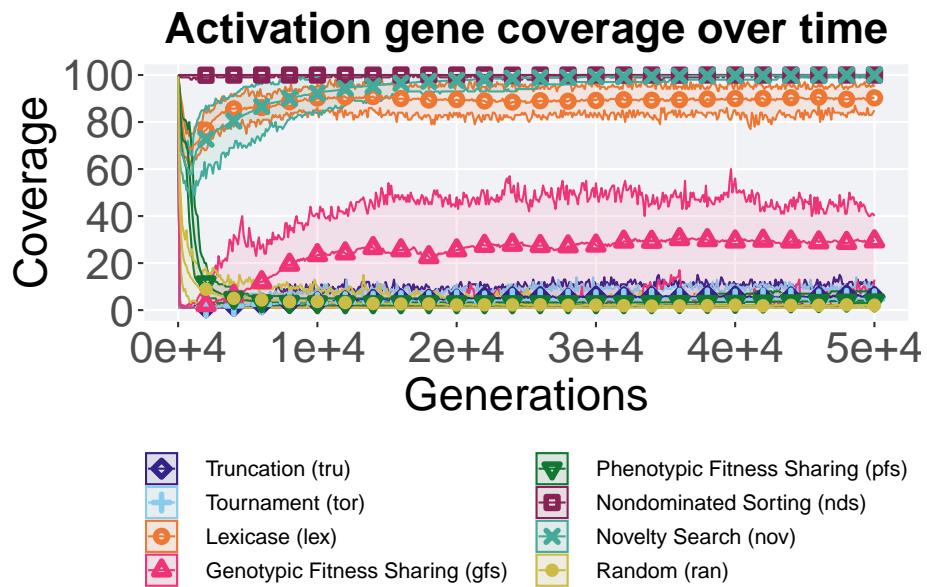
## `summarise()` has grouped output by 'Selection Scheme'. You can override using
## the ` .groups` argument.

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.

```

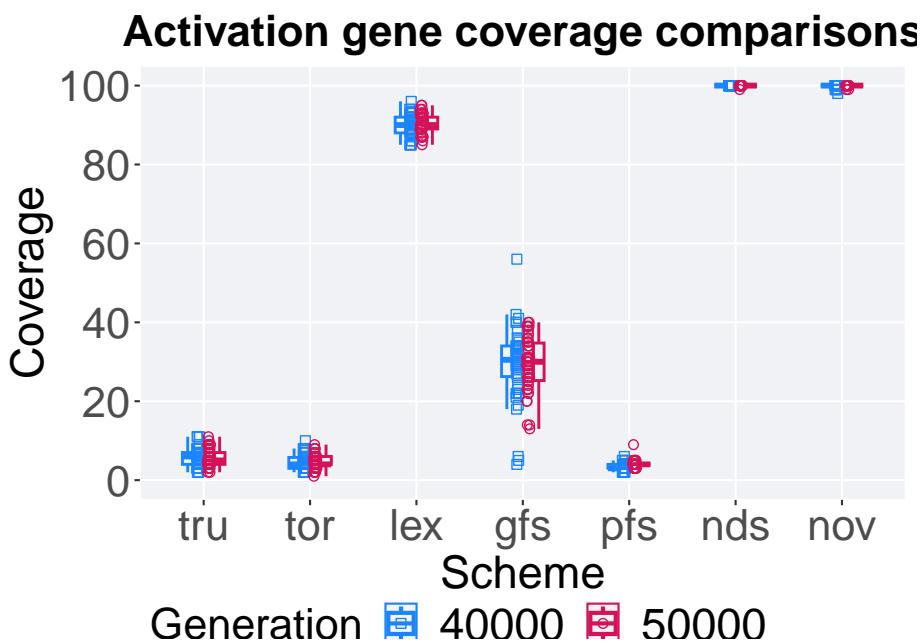
mvc_p = ggplot(mid, aes(x = acron, y=uni_str_pos, group = acron, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), lwd = 1, alpha = 0.5) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], outlier.colour = "black") +
  geom_point(data = end, aes(x = acron, y=uni_str_pos), col = mvc_col[2], position = position_nudge(x = 0.15), lwd = 1, alpha = 0.5) +
  geom_boxplot(data = end, aes(x = acron, y=uni_str_pos), position = position_nudge(x = 0.15), lwd = 0.7, col = mvc_col[2], outlier.colour = "black") +
  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(),
    mean_coverage = mean(cov),
    median_coverage = median(cov),
    min_coverage = min(cov),
    max_coverage = max(cov)
  )

```

```

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>   <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(
## 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(PupilometryR)
library(sdamr)
```

5.2 Performance

Performance analysis.

5.2.1 Over time

Best performance in a population over time.

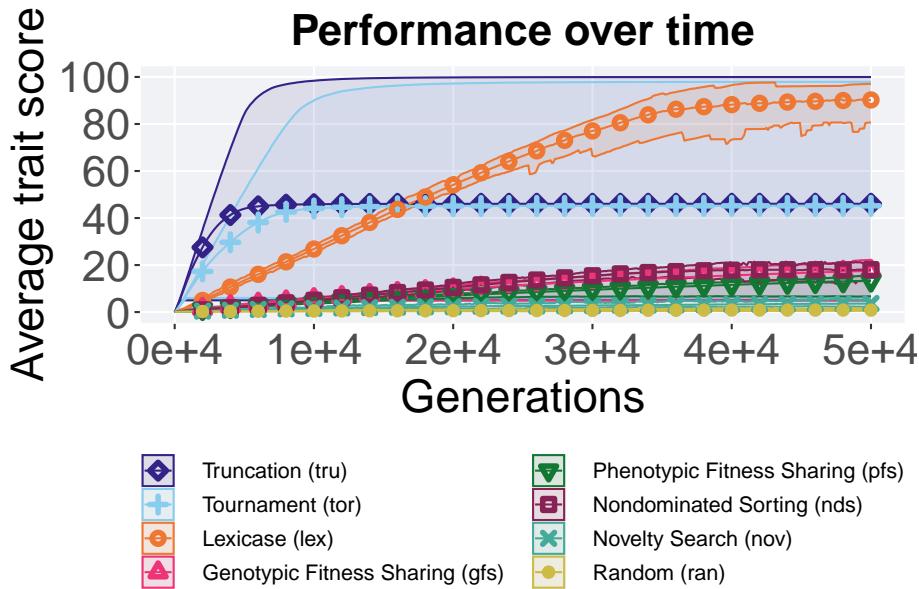
```

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

```

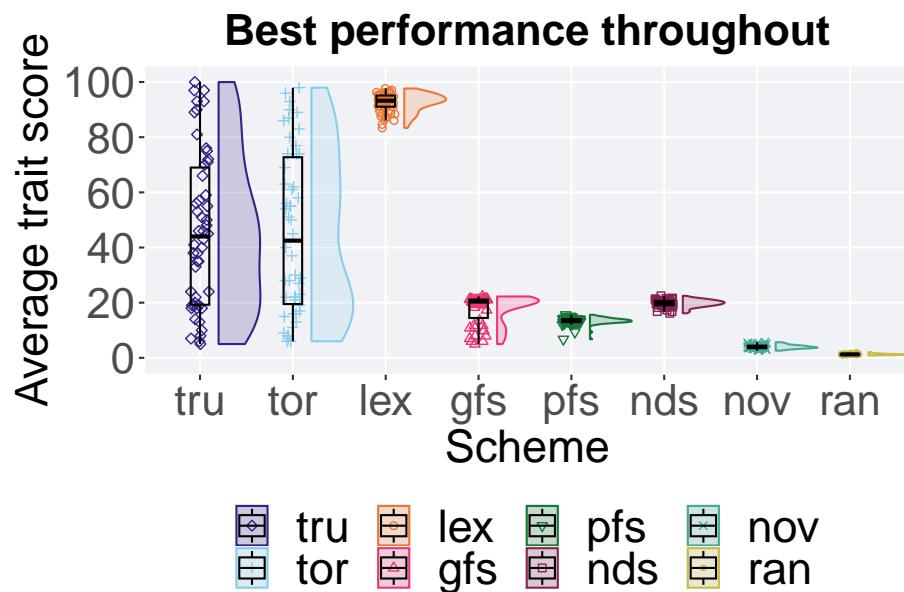


5.2.2 Best performance throughout

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_exploration')
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.

```
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.

```
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 -
## ran < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
##
## P value adjustment method: bonferroni
```

5.2.3 End of 50,000 generations

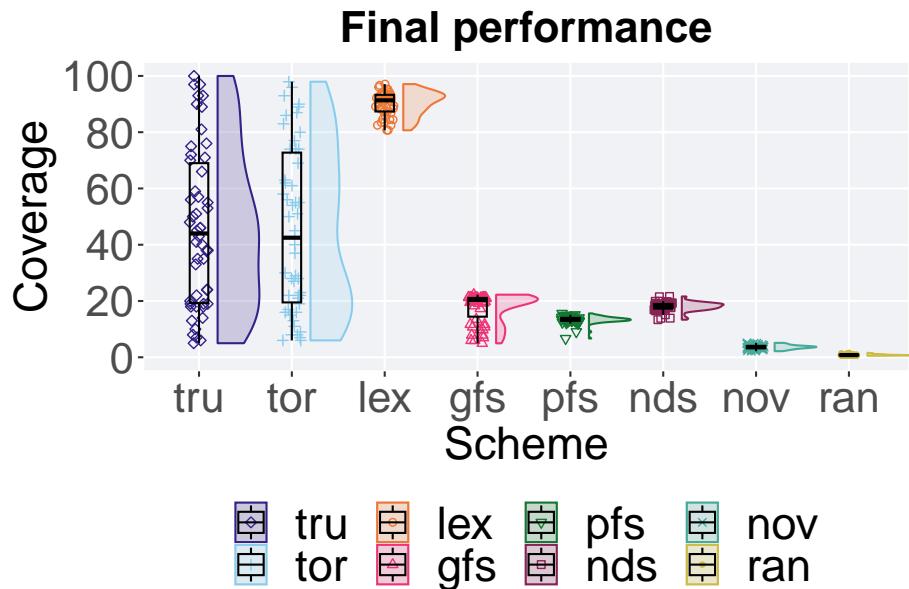
Best performance in the population at the end of 50,000 generations.

```
# end of run
filter(cc_over_time, diagnostic == 'multipath_exploration' & gen == 50000) %>%
```

```

ggplot(., aes(x = acron, y = pop_fit_max / DIMENSIONALITY, color = acron, fill = acron)
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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> <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 = "bonf"
                      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.

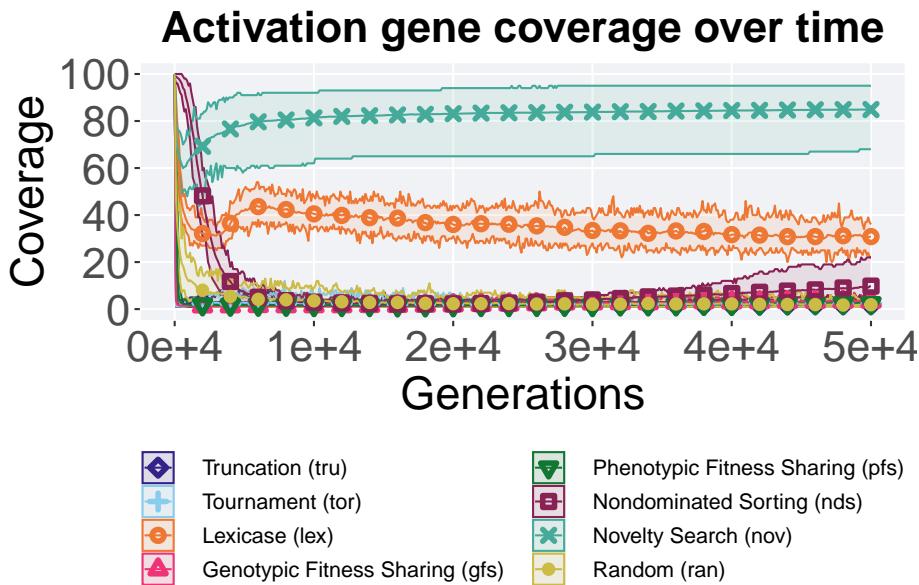
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill = `Selection\nScheme`))
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2)
  scale_y_continuous(
    name="Coverage",
    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.

```

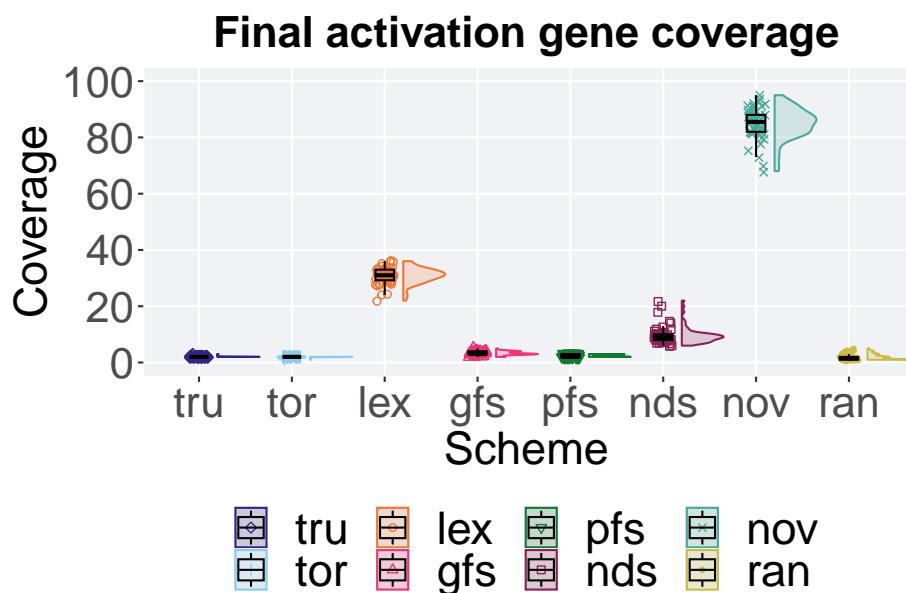
# end of run
filter(cc_over_time, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = acron, y = uni_str_pos, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.2) +
  geom_point(position = position_jitter(width = .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> <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 = '1')

##
## 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.

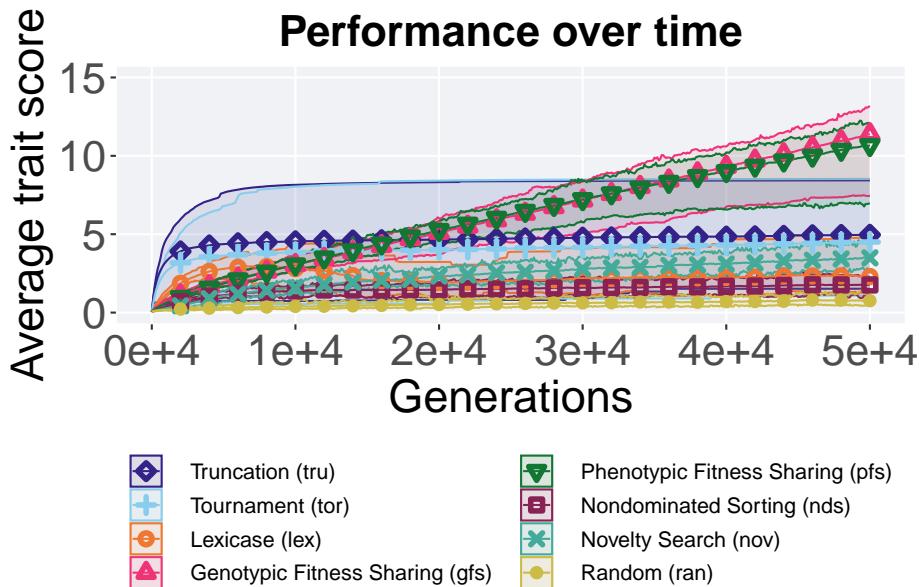
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill = `Selection\nScheme`,
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2)
  scale_y_continuous(
    name="Average 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.

```

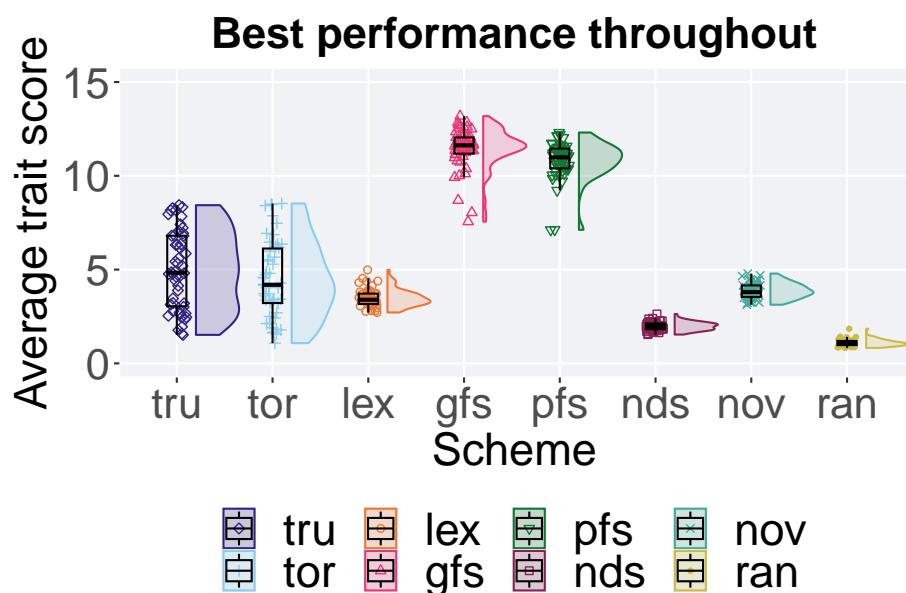
### best performance throughout
filter(cc_best_mvc, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = acron, y = val / DIMENSIONALITY, color = acron, fill = acron, shape = acron)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 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 &
end$Generation <- factor(end$gen)

mid = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & gen == 40000 &
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, nudge_x = -.15), size = 100),
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], width = 0.5))

  geom_point(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[1]),
  geom_boxplot(data = end, aes(x = acron, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 0), col = mvc_col[1], width = 0.5))

  scale_y_continuous(
    name="Average trait score",
    limits=c(0, 15),
    breaks=seq(0,15, 5),
    labels=c("0", "5", "10", "15"))
) +

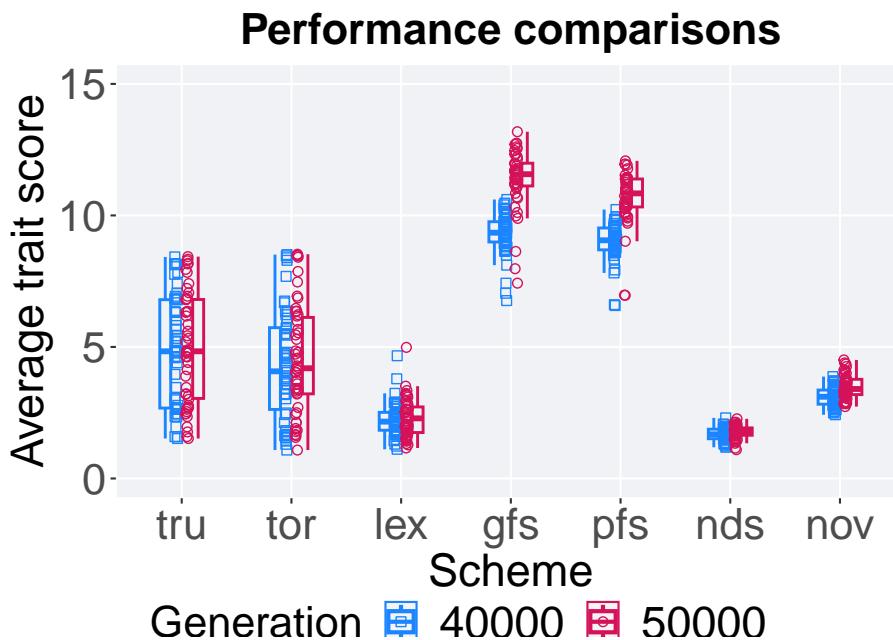
```

```

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>
## 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.

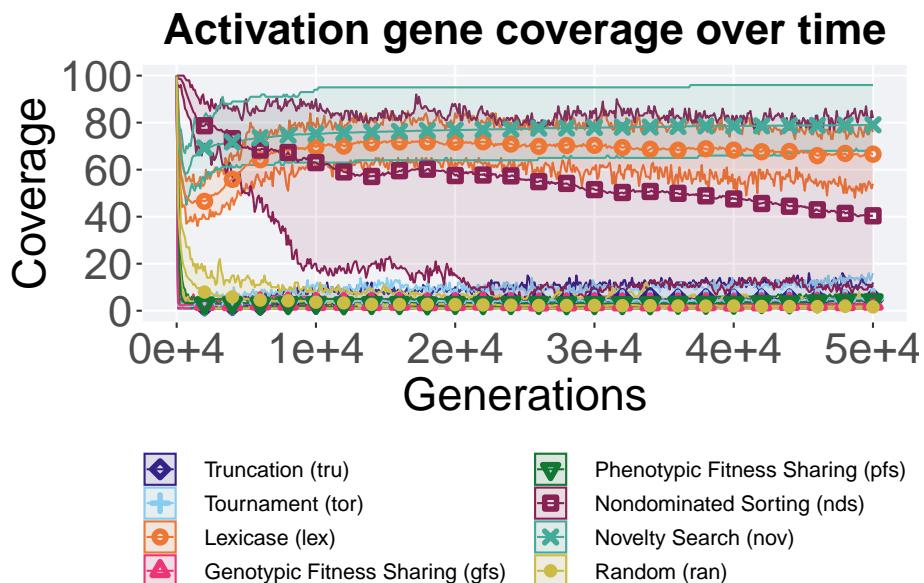
ggplot(lines, aes(x=gen, y=mean, group = `Selection\nScheme`, fill = `Selection\nScheme`))
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +

```

```

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

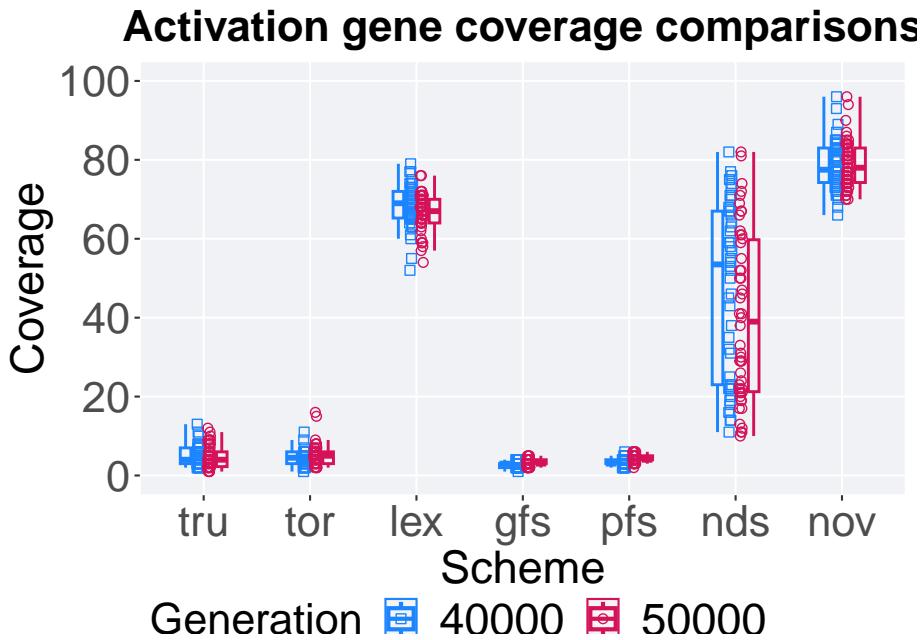
Best activation gene coverage in the population at 40,000 and 50,000 generations.

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

mid = filter(cc_over_time_mvc, diagnostic == 'multipath_exploration' & gen == 40000 &
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, nudge_x = -.17), size = 10) +
  geom_boxplot(position = position_nudge(x = -.17, y = 0), lwd = 0.7, col = mvc_col[1], width = 0.5) +
  geom_point(data = end, aes(x = acron, y=uni_str_pos), col = mvc_col[2], position = position_nudge(x = 0, y = 0), size = 10) +
  geom_boxplot(data = end, aes(x = acron, y=uni_str_pos), position = position_nudge(x = 0, y = 0), lwd = 0.7, col = mvc_col[2], width = 0.5) +
  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 == 40000))
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>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

##   <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
```

Chapter 6

Truncation selection

We present the results from our parameter sweep on truncation selection. 50 replicates are conducted for each truncation size T parameter value explored.

```
library(ggplot2)
library(cowplot)
library(dplyr)
library(PupilometryR)
```

6.1 Exploitation rate results

Here we present the results for **best performances** found by each truncation selection value replicate on the exploitation rate diagnostic.

6.1.1 Performance over time

Performance over time.

```
lines = filter(tru_ot, diagnostic == 'exploitation_rate') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

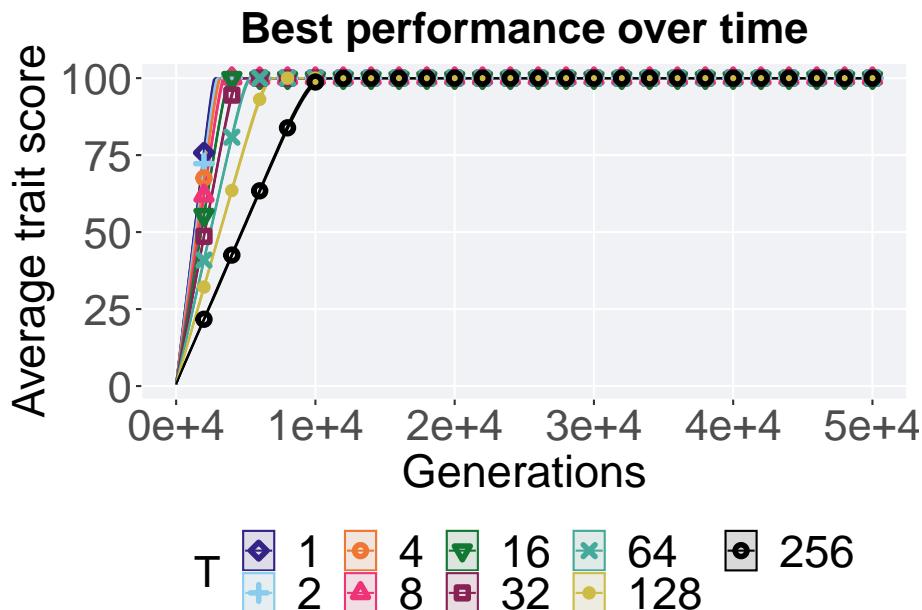
## `summarise()` has grouped output by 'T'. You can override using the `.`groups` argument.

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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"
) +
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("Best performance over time") +
p_theme

```



6.1.2 Generation satisfactory solution found

The first Generations a satisfactory solution is found throughout the 50,000 generations.

```

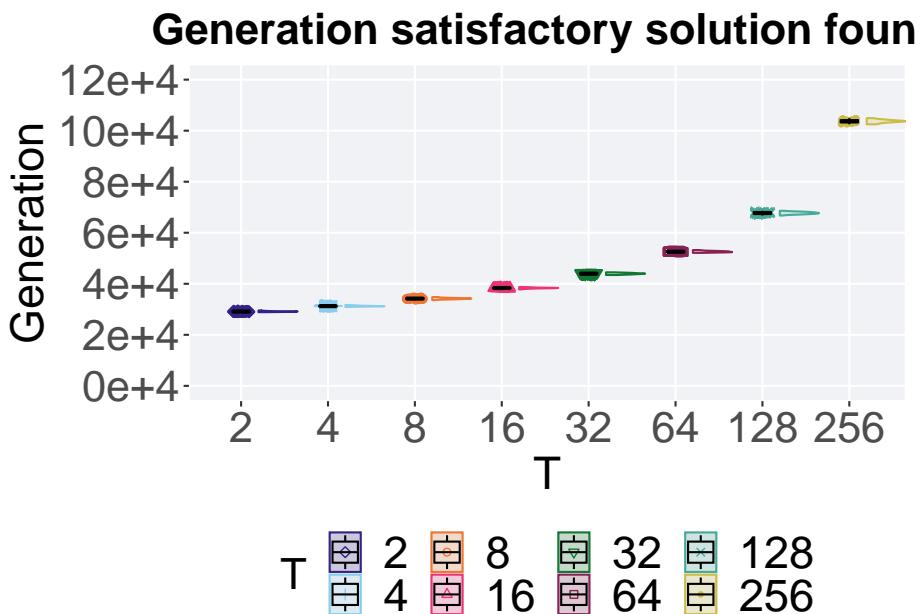
filter(tru_ssf, Diagnostic == 'EXPLOITATION_RATE') %>%
ggplot(., aes(x = T, y = Generations, color = T, fill = T, shape = T)) +

```

```

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_shape_manual(values=SHAPE) +
  scale_y_continuous(
    name="Generation",
    limits=c(0, 12000),
    breaks=c(0, 2000, 4000, 6000, 8000, 10000, 12000),
    labels=c("0e+4", "2e+4", "4e+4", "6e+4", "8e+4", "10e+4", "12e+4")
  ) +
  scale_x_discrete(
    name="T"
  ) +
  ggtitle("Generation satisfactory solution found") +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  p_theme

```



6.1.2.1 Stats

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

```

ssf = filter(tru_ssf, Diagnostic == 'EXPLOITATION_RATE')
group_by(ssf, T) %>%

```

```
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: 8 x 8
##   T      count na_cnt   min median   mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <int> <dbl>
## 1 2       50     0  2887  2912. 2912. 2955  18.2
## 2 4       50     0  3091  3125  3126. 3171  19
## 3 8       50     0  3357  3420  3421. 3481  34.2
## 4 16      50     0  3781  3834. 3833. 3873  20.8
## 5 32      50     0  4344  4396. 4396. 4450  41.2
## 6 64      50     0  5211  5256. 5259. 5322  38
## 7 128     50     0  6675  6773  6772. 6861  62
## 8 256     50     0 10250 10368. 10369. 10492 73.2
```

Kruskal–Wallis test provides evidence of significant differences among the Generations a satisfactory solution is first found.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: Generations by T
## 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 on the Generations a satisfactory solution is first found. .

```
pairwise.wilcox.test(x = ssf$Generations, g = ssf$T, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 'g')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: ssf$Generations and ssf$T
##
##      2      4      8      16      32      64     128
## 4 <2e-16 -      -      -      -      -      -
## 8 <2e-16 <2e-16 -      -      -      -      -
## 16 <2e-16 <2e-16 <2e-16 -      -      -      -
```

```
## 32 <2e-16 <2e-16 <2e-16 <2e-16 - - -
## 64 <2e-16 <2e-16 <2e-16 <2e-16 - - -
## 128 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 - -
## 256 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

6.1.3 Multi-valley crossing

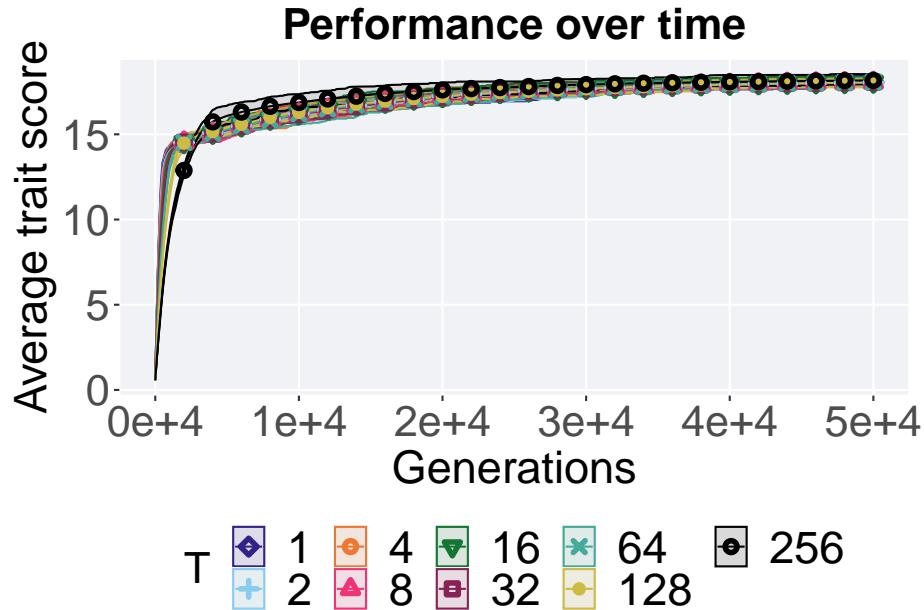
6.1.3.1 Performance over time

```
# data for lines and shading on plots
lines = filter(tru_ot_mvc, diagnostic == 'exploitation_rate') %>%
  group_by(T, 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 'T'. You can override using the `.`groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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="Average trait score"
  ) +
  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 +
  guides(
    shape=guide_legend(nrow=2, title.position = "left"),
    color=guide_legend(nrow=2, title.position = "left"),
    fill=guide_legend(nrow=2, title.position = "left")
  )
```



6.1.3.2 Performance comparison

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

```
# 80% and final generation comparison
end = filter(tru_ot_mvc, diagnostic == 'exploitation_rate' & gen == 50000 & T != 'ran')
end$Generation <- factor(end$gen)

mid = filter(tru_ot_mvc, diagnostic == 'exploitation_rate' & gen == 40000 & T != 'ran')
mid$Generation <- factor(mid$gen)

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

  geom_point(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2])
  geom_boxplot(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="T"
  ) +
```

```

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,  

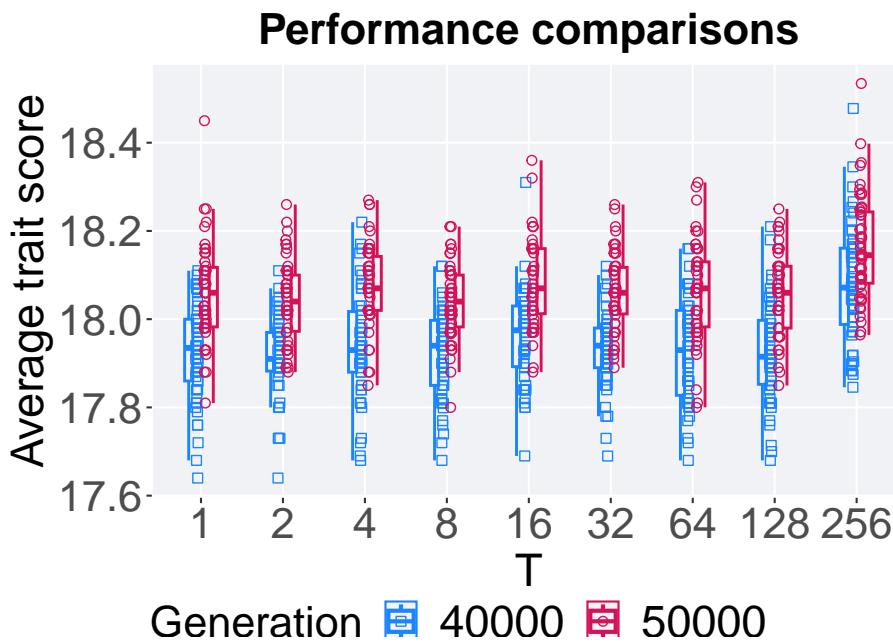
  nrow=2,  

  rel_heights = c(1,.05),  

  label_size = TSIZE  

)

```



6.1.3.3 Stats

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

```

slices = filter(tru_ot_mvc, diagnostic == 'exploitation_rate' & (gen == 50000 | gen == 40000))  

slices$Generation <- factor(slices$gen, levels = c(50000,40000))  

slices %>%  

  group_by(T, 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 'T'. You can override using the `groups`#
## argument.

## # A tibble: 18 x 9
## # Groups:   T [9]
##   T     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>      <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1      50000       50    0  17.8  18.1  18.1  18.4  0.135
## 2 1      40000       50    0  17.6  17.9  17.9  18.1  0.140
## 3 2      50000       50    0  17.9  18.0  18.0  18.3  0.127
## 4 2      40000       50    0  17.6  17.9  17.9  18.1  0.0875
## 5 4      50000       50    0  17.8  18.1  18.1  18.3  0.122
## 6 4      40000       50    0  17.7  17.9  17.9  18.2  0.138
## 7 8      50000       50    0  17.8  18.0  18.0  18.2  0.118
## 8 8      40000       50    0  17.7  17.9  17.9  18.1  0.147
## 9 16     50000       50    0  17.9  18.1  18.1  18.4  0.148
## 10 16    40000       50    0  17.7  18.0  18.0  18.3  0.137
## 11 32     50000       50    0  17.9  18.1  18.1  18.3  0.106
## 12 32     40000       50    0  17.7  17.9  17.9  18.1  0.0900
## 13 64     50000       50    0  17.8  18.1  18.1  18.3  0.147
## 14 64     40000       50    0  17.7  17.9  17.9  18.2  0.192
## 15 128    50000       50    0  17.8  18.1  18.1  18.2  0.140
## 16 128    40000       50    0  17.7  17.9  17.9  18.2  0.145
## 17 256    50000       50    0  18.0  18.1  18.2  18.5  0.162
## 18 256    40000       50    0  17.8  18.1  18.1  18.5  0.173

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 2109.5, p-value = 3.13e-09
## alternative hypothesis: true location shift is not equal to 0

T 4

```

```
wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 4 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_fit_max and filter(slices, T == 4 & Ge
## W = 2003.5, p-value = 2.067e-07
## alternative hypothesis: true location shift is not equal to 0

T 8
wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 8 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_fit_max and filter(slices, T == 8 & Ge
## W = 2037.5, p-value = 5.705e-08
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 16 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_fit_max and filter(slices, T == 16 &
## W = 1998.5, p-value = 2.457e-07
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 32 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_fit_max and filter(slices, T == 32 &
## W = 2151, p-value = 5.311e-10
## alternative hypothesis: true location shift is not equal to 0
```

T 64

```
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 64 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1997, p-value = 2.628e-07
## alternative hypothesis: true location shift is not equal to 0
```

T 128

```
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 128 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 2022.5, p-value = 1.009e-07
## alternative hypothesis: true location shift is not equal to 0
```

T 256

```
wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 256 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1742, p-value = 0.0007032
## alternative hypothesis: true location shift is not equal to 0
```

6.2 Ordered exploitation results

Here we present the results for **best performances** found by each truncation selection size value 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 and 100.

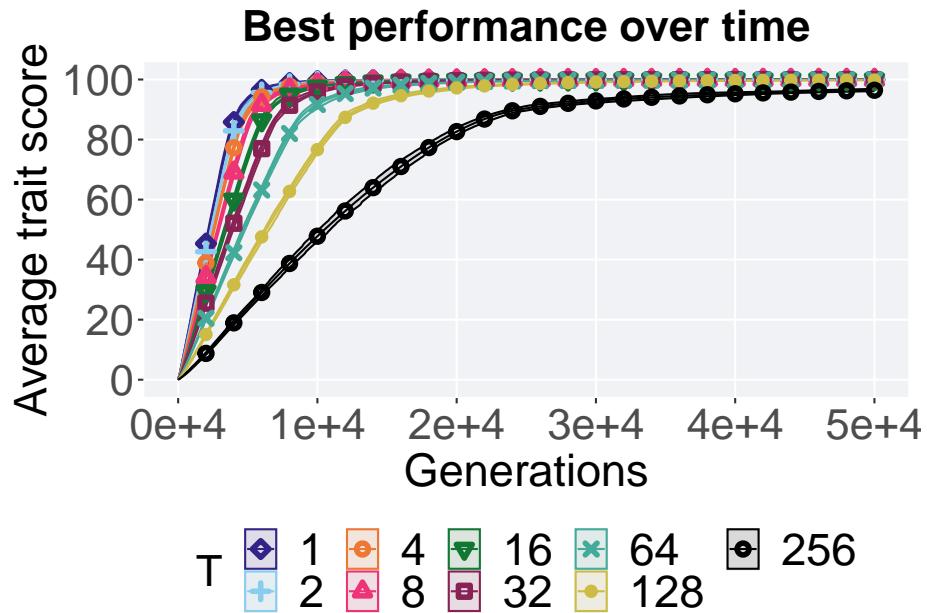
6.2.1 Performance over time

Performance over time.

```
lines = filter(tru_ot, diagnostic == 'ordered_exploitation') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

## `summarise()` has grouped output by 'T'. You can override using the ` `.groups` ` argument.

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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="Average trait score",
    limits=c(-1, 101),
    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("Best performance over time") +
  p_theme
```

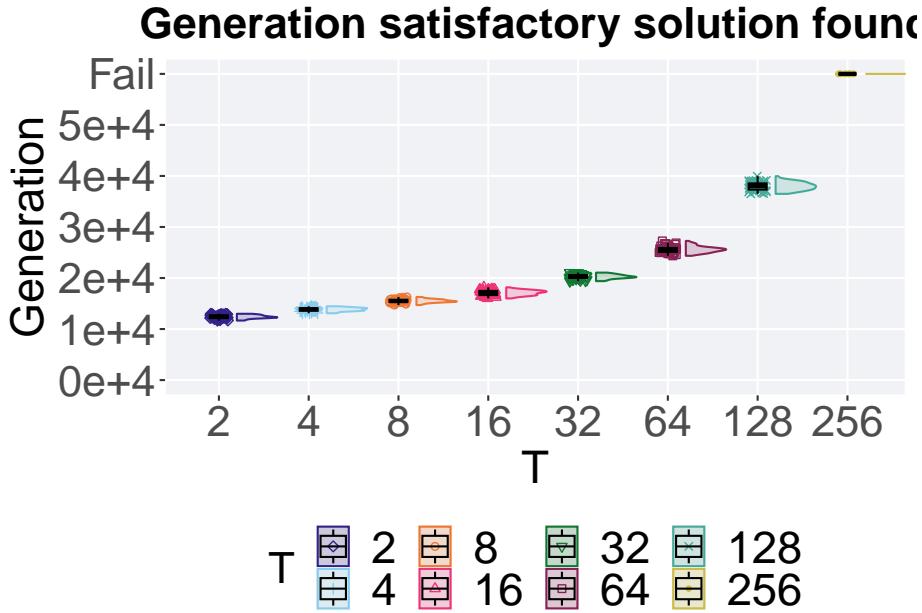


6.2.2 Generation satisfactory solution found

The first Generations a satisfactory solution is found throughout the 50,000 generations.

```
filter(tru_ssf, Diagnostic == 'ORDERED_EXPLOITATION') %>%
  ggplot(., aes(x = T, y = Generations, color = T, fill = T, shape = T)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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_shape_manual(values=SHAPE) +
  scale_y_continuous(
    name="Generation",
    limits=c(0, 60000),
    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="T"
  ) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Generation satisfactory solution found") +
  p_theme
```

```
## Warning: Removed 34 rows containing missing values (`geom_point()`).
```



6.2.2.1 Stats

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

```
ssf = filter(ssf, Diagnostic == 'ORDERED_EXPLOITATION')
group_by(ssf, T) %>%
  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: 8 x 8
##   T      count na_cnt  min median   mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <int> <dbl>
## 1 2       50      0 11664 12356. 12407. 12992  496.
## 2 4       50      0 13096 13871  13840. 14459  496.
## 3 8       50      0 14701 15466. 15511. 16280  422.
## 4 16      50      0 16002 17192  17098. 18174  813.
```

```
## 5 32      50      0 19392 20223 20274. 21055 555.
## 6 64      50      0 24348 25568. 25598. 27268 764.
## 7 128     50      0 36490 38016 37967. 39959 1210.
## 8 256     50      0 60000 60000 60000 60000     0
```

Kruskal–Wallis test provides evidence of significant differences among the first Generations a satisfactory solution is found throughout the 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: Generations by T
## Kruskal-Wallis chi-squared = 393.49, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the first Generations a satisfactory solution is found throughout the 50,000 generations.

```
pairwise.wilcox.test(x = ssf$Generations, g = ssf$T , p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 'g')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: ssf$Generations and ssf$T
##
##      2      4      8      16      32      64     128
## 4 <2e-16 -      -      -      -      -      -
## 8 <2e-16 <2e-16 -      -      -      -      -
## 16 <2e-16 <2e-16 <2e-16 -      -      -      -
## 32 <2e-16 <2e-16 <2e-16 <2e-16 -      -      -
## 64 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -      -
## 128 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -
## 256 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

6.2.3 Multi-valley crossing

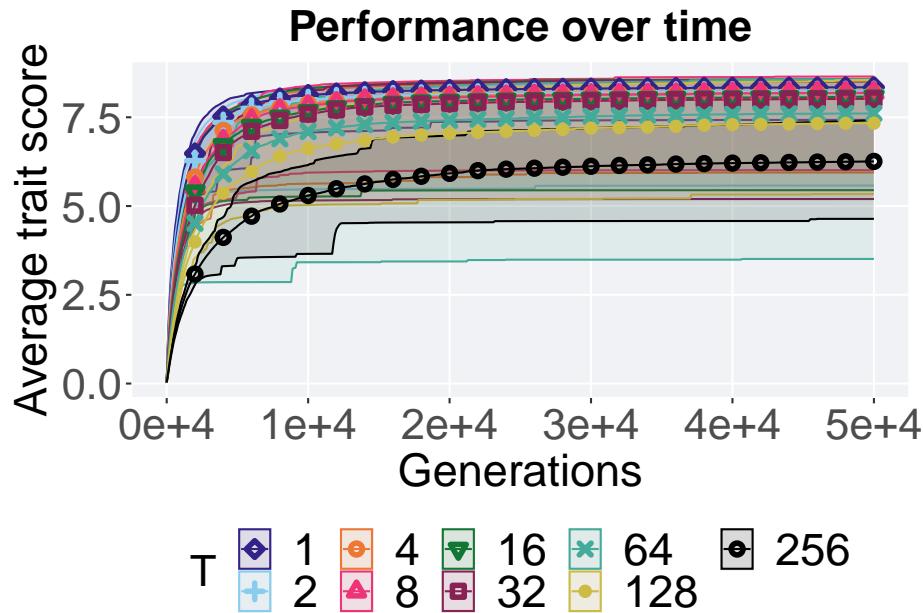
6.2.3.1 Performance over time

```
# data for lines and shading on plots
lines = filter(tru_ot_mvc, diagnostic == 'ordered_exploitation') %>%
  group_by(T, 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 'T'. You can override using the `groups` argument.
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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="Average trait score"
    ) +
    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 +
    guides(
      shape=guide_legend(nrow=2, title.position = "left"),
      color=guide_legend(nrow=2, title.position = "left"),
      fill=guide_legend(nrow=2, title.position = "left")
    )
}
```



6.2.3.2 Performance comparison

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

```
# 80% and final generation comparison
end = filter(tru_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 50000 & T != 'random')
end$Generation <- factor(end$gen)

mid = filter(tru_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 40000 & T != 'random')
mid$Generation <- factor(mid$gen)

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

  geom_point(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], size = 1)
  geom_boxplot(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 1), lwd = 0.7, col = mvc_col[2])

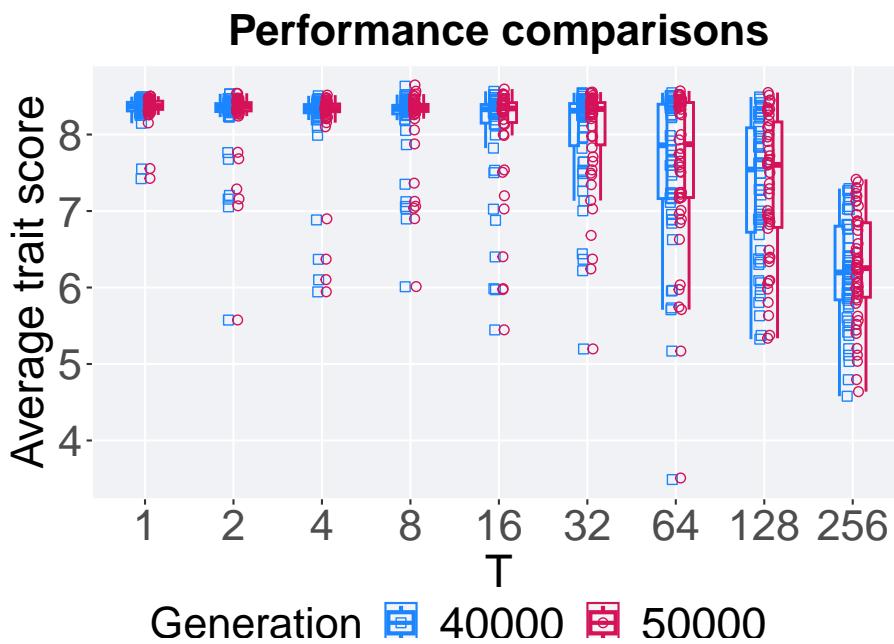
  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="T"
  ) +
  scale_shape_manual(values=c(0,1)) +
```

```

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
)

```



6.2.3.3 Stats

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

```

slices = filter(tru_ot_mvc, diagnostic == 'ordered_exploitation' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max / DIMENSIONALITY)),
    mean = mean(pop_fit_max / DIMENSIONALITY),
    min = min(pop_fit_max / DIMENSIONALITY),
    max = max(pop_fit_max / DIMENSIONALITY),
    median = median(pop_fit_max / DIMENSIONALITY),
    stdev = sd(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 'T'. You can override using the `groups` argument.

## # A tibble: 18 x 9
## # Groups:   T [9]
##   T     Generation count na_cnt   min median   mean   max    IQR
##   <fct> <fct>      <int> <dbl> <dbl> <dbl> <dbl> <dbl>
##   1 1       50000      50    0  7.43  8.37  8.34  8.50  0.108
##   2 1       40000      50    0  7.42  8.37  8.33  8.50  0.113
##   3 2       50000      50    0  5.58  8.36  8.23  8.54  0.108
##   4 2       40000      50    0  5.57  8.36  8.21  8.53  0.104
##   5 4       50000      50    0  5.94  8.35  8.19  8.52  0.102
##   6 4       40000      50    0  5.94  8.33  8.18  8.51  0.107
##   7 8       50000      50    0  6.01  8.35  8.19  8.65  0.0922
##   8 8       40000      50    0  6.01  8.33  8.17  8.63  0.112
##   9 16      50000      50    0  5.45  8.34  8.08  8.59  0.260
##  10 16     40000      50    0  5.45  8.33  8.05  8.56  0.244
##  11 32      50000      50    0  5.20  8.33  8.02  8.56  0.553
##  12 32     40000      50    0  5.20  8.31  8.00  8.55  0.551
##  13 64      50000      50    0  3.51  7.87  7.61  8.57  1.24
##  14 64      40000      50    0  3.49  7.86  7.58  8.55  1.23
##  15 128     50000      50    0  5.34  7.60  7.33  8.55  1.38
##  16 128     40000      50    0  5.32  7.54  7.29  8.49  1.37
##  17 256     50000      50    0  4.64  6.25  6.26  7.41  0.973
##  18 256     40000      50    0  4.58  6.20  6.21  7.29  0.963

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1359, p-value = 0.4545
## alternative hypothesis: true location shift is not equal to 0

T 4

```

```
wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 4 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_fit_max and filter(slices, T == 4 & Ge
## W = 1355, p-value = 0.4713
## alternative hypothesis: true location shift is not equal to 0

T 8

wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 8 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_fit_max and filter(slices, T == 8 & Ge
## W = 1375, p-value = 0.3907
## alternative hypothesis: true location shift is not equal to 0

T 16

wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 16 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_fit_max and filter(slices, T == 16 &
## W = 1367, p-value = 0.4219
## alternative hypothesis: true location shift is not equal to 0

T 32

wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 32 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_fit_max and filter(slices, T == 32 &
## W = 1320, p-value = 0.6319
## alternative hypothesis: true location shift is not equal to 0
```

T 64

```
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 64 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1319, p-value = 0.6368
## alternative hypothesis: true location shift is not equal to 0
```

T 128

```
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 128 & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1311, p-value = 0.6766
## alternative hypothesis: true location shift is not equal to 0
```

T 256

```
wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 256 & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1321, p-value = 0.627
## alternative hypothesis: true location shift is not equal to 0
```

6.3 Contradictory objectives diagnostic

Here we present the results for **satisfactory trait coverage** and **activation gene coverage** found by each truncation selection size value replicate on the ordered exploitation diagnostic. Satisfactory trait coverage refers to the count of unique satisfied traits in the population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both coverage values fall between 0 and 100.

6.3.1 Satisfactory trait coverage

Satisfactory trait coverage analysis.

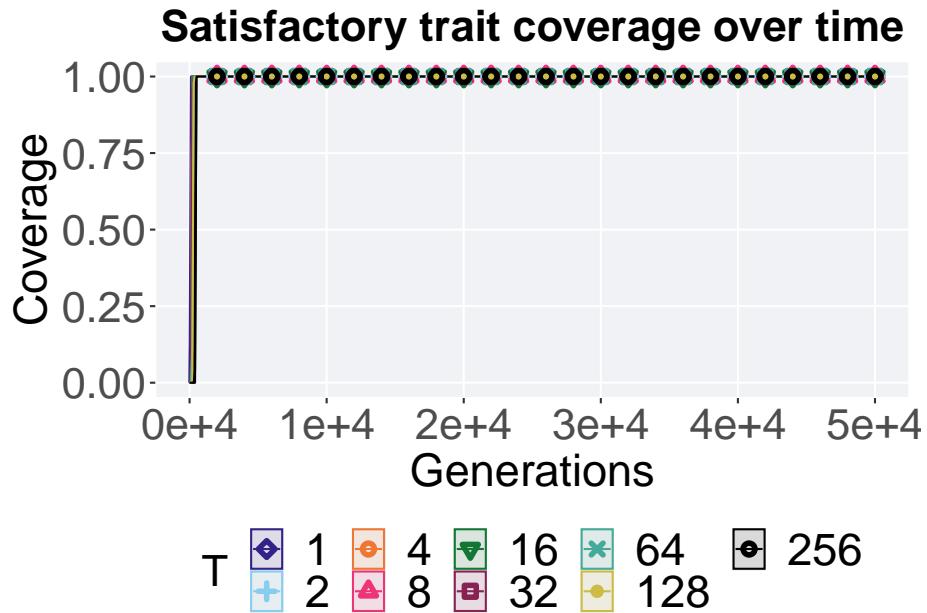
6.3.1.1 Coverage over time

Satisfactory trait coverage over time.

```
lines = filter(tru_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

## `summarise()` has grouped output by 'T'. You can override using the `groups` argument.

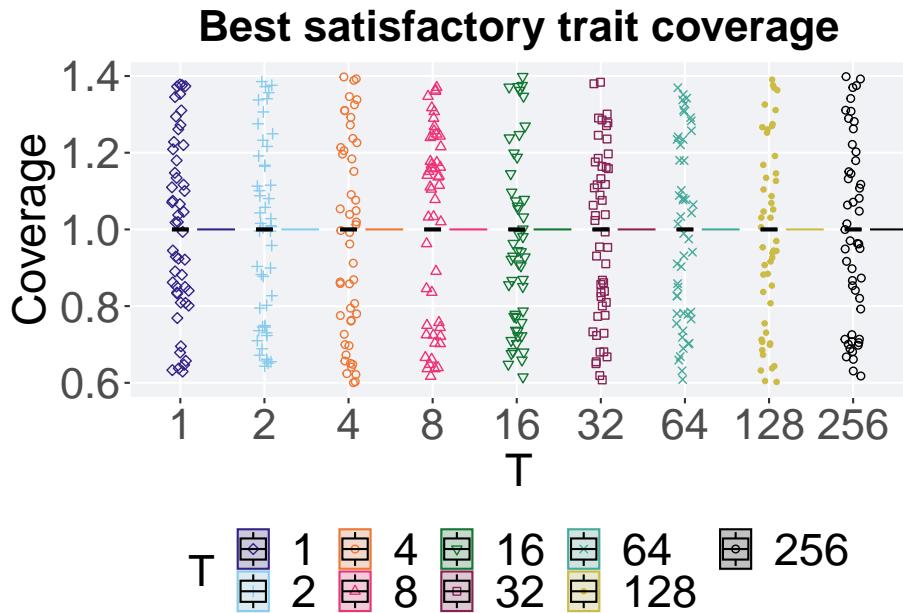
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage"
  ) +
  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
```



6.3.1.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
filter(tru_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = T, y = val, color = T, fill = T, shape = T)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.0) +
  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"
  ) +
  scale_x_discrete(
    name="T"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best satisfactory trait coverage") +
  p_theme
```



6.3.1.2.1 Stats

Summary statistics for the best satisfactory trait coverage throughout 50,000 generations.

```
coverage = filter(tru_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives')
group_by(coverage, T) %>%
  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)
  )

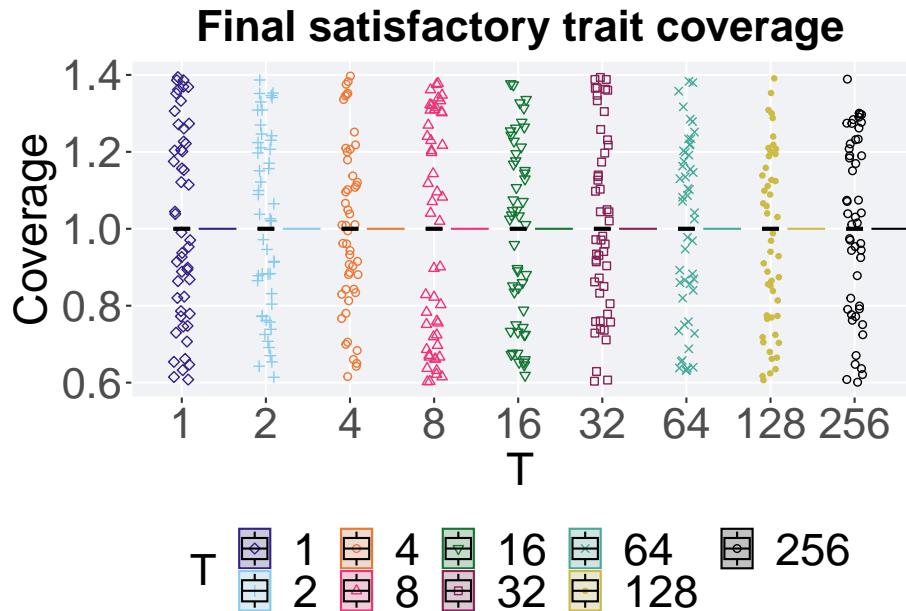
## # A tibble: 9 x 8
##   T      count na_cnt  min median  mean  max  IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1       50     0     1     1     1     1     0
## 2 2       50     0     1     1     1     1     0
## 3 4       50     0     1     1     1     1     0
## 4 8       50     0     1     1     1     1     0
## 5 16      50     0     1     1     1     1     0
## 6 32      50     0     1     1     1     1     0
```

```
## 7 64      50     0     1     1     1     1     0
## 8 128     50     0     1     1     1     1     0
## 9 256     50     0     1     1     1     1     0
```

6.3.1.3 End of 50,000 generations

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

```
filter(tru_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = T, y = pop_uni_obj, color = T, fill = T, shape = T)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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"
  ) +
  scale_x_discrete(
    name = "T"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final satisfactory trait coverage") +
  p_theme
```



6.3.1.3.1 Stats

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

```
coverage = filter(tru_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, T) %>%
  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: 9 x 8
##   T      count na_cnt  min median  mean  max  IQR
##   <fct> <int> <int> <dbl> <dbl> <int> <dbl>
## 1 1       50     0     1     1     1     1     0
## 2 2       50     0     1     1     1     1     0
## 3 4       50     0     1     1     1     1     0
## 4 8       50     0     1     1     1     1     0
## 5 16      50     0     1     1     1     1     0
## 6 32      50     0     1     1     1     1     0
## 7 64      50     0     1     1     1     1     0
## 8 128     50     0     1     1     1     1     0
## 9 256     50     0     1     1     1     1     0
```

6.3.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the contradictory objectives diagnostic.

6.3.2.1 Coverage over time

Activation gene coverage over time.

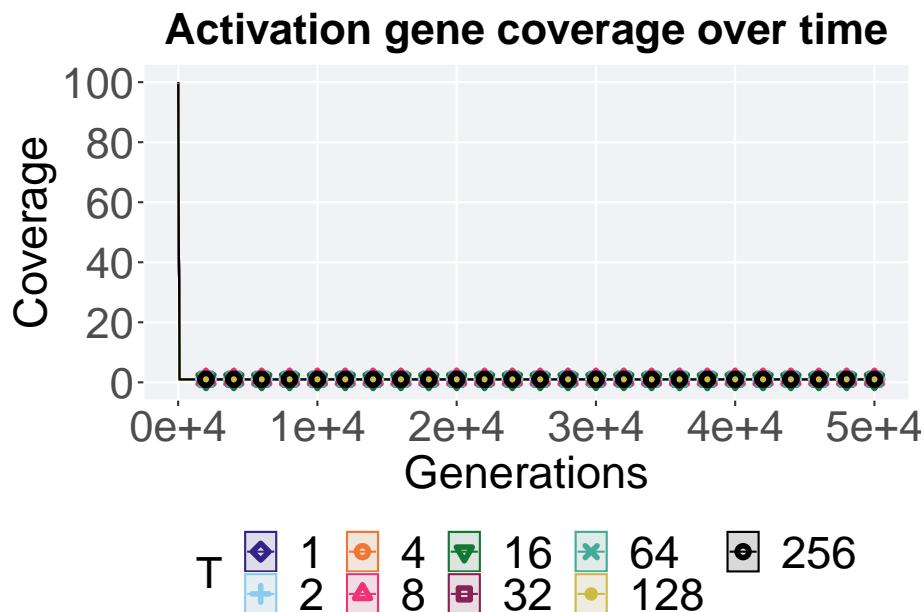
```
lines = filter(tru_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

## `summarise()` has grouped output by 'T'. You can override using the `.groups` argument.
```

```

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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(-1, 101),
    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

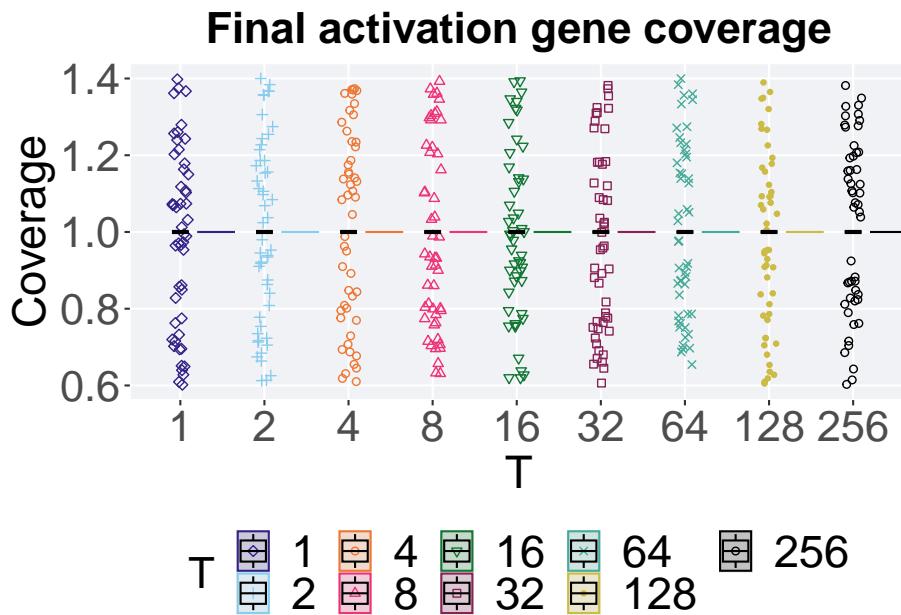
```



6.3.2.2 End of 50,000 generations

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

```
filter(tru_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = T, y = uni_str_pos, color = T, fill = T, shape = T)) +
  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"
  ) +
  scale_x_discrete(
    name = "T"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final activation gene coverage") +
  p_theme
```



6.3.2.2.1 Stats

Summary statistics for activation gene coverage in the population at the end of 50,000 generations.

```

coverage = filter(tru_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, T) %>%
  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: 9 x 8
##   T     count na_cnt  min median  mean  max  IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 1      50      0     1     1     1     1     0
## 2 2      50      0     1     1     1     1     0
## 3 4      50      0     1     1     1     1     0
## 4 8      50      0     1     1     1     1     0
## 5 16     50      0     1     1     1     1     0
## 6 32     50      0     1     1     1     1     0
## 7 64     50      0     1     1     1     1     0
## 8 128    50      0     1     1     1     1     0
## 9 256    50      0     1     1     1     1     0

```

6.3.3 Multi-valley crossing

6.3.3.1 Satisfactory trait coverage over time

```

lines = filter(tru_ot_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

## `summarise()` has grouped output by 'T'. You can override using the `.groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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"

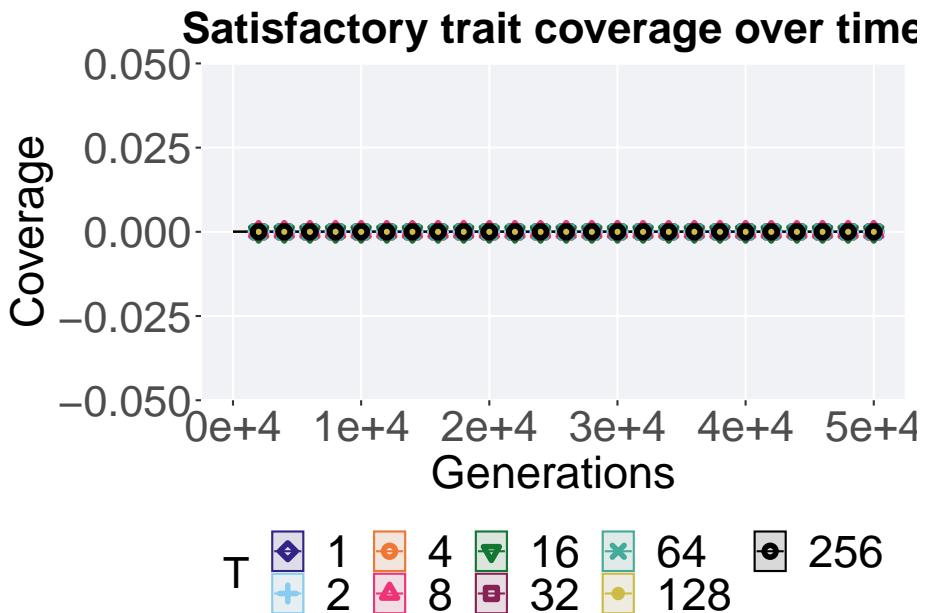
```

```

) +
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

```



6.3.3.2 Satisfactory trait coverage comparison

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

```

# 80% and final generation comparison
end = filter(tru_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 50000)
end$Generation <- factor(end$gen)

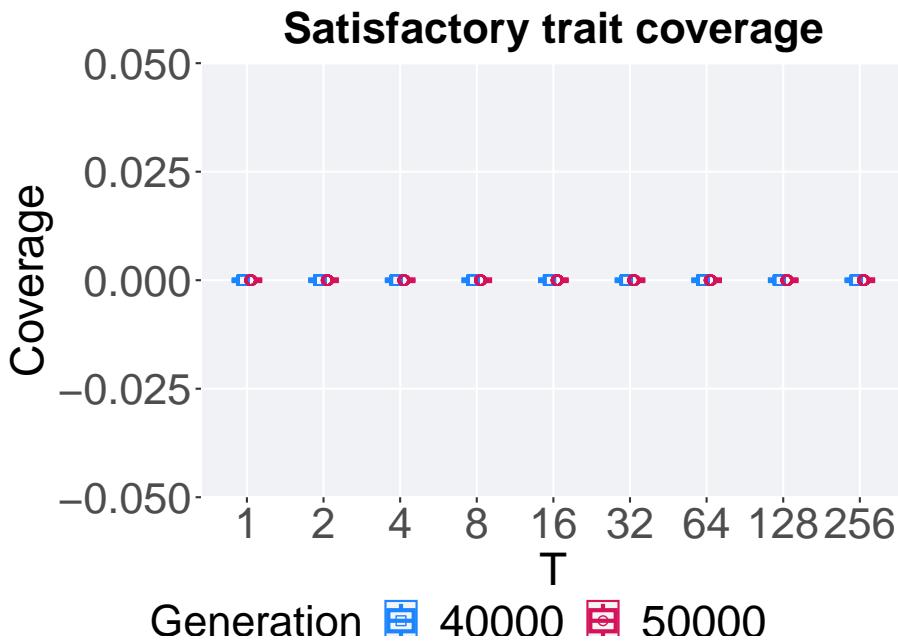
mid = filter(tru_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 40000)
mid$Generation <- factor(mid$gen)

```

```
mvc_p = ggplot(mid, aes(x = T, y=pop_uni_obj, group = T, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), lwd = 0.7, col = mvc_col[1])
  geom_point(data = end, aes(x = T, y=pop_uni_obj), col = mvc_col[2], position = position_nudge(x = .15))
  geom_boxplot(data = end, aes(x = T, y=pop_uni_obj), position = position_nudge(x = .15))

  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="T"
  ) +
  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") +
      theme(legend.position="none"),
    legend,
    nrow=2,
    rel_heights = c(1,.05),
    label_size = TSIZE
  )
)
```



6.3.3.2.1 Stats

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

```
slices = filter(tru_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, 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 'T'. You can override using the `.groups` argument.

## # A tibble: 18 x 9
## # Groups:   T [9]
##   T     Generation count na_cnt   min median   mean   max    IQR
##   <fct> <fct>     <int> <int> <int> <dbl> <dbl> <int> <dbl>
```

```

## 1 1      50000      50      0      0      0      0      0      0
## 2 1      40000      50      0      0      0      0      0      0
## 3 2      50000      50      0      0      0      0      0      0
## 4 2      40000      50      0      0      0      0      0      0
## 5 4      50000      50      0      0      0      0      0      0
## 6 4      40000      50      0      0      0      0      0      0
## 7 8      50000      50      0      0      0      0      0      0
## 8 8      40000      50      0      0      0      0      0      0
## 9 16     50000      50      0      0      0      0      0      0
## 10 16    40000      50      0      0      0      0      0      0
## 11 32    50000      50      0      0      0      0      0      0
## 12 32    40000      50      0      0      0      0      0      0
## 13 64    50000      50      0      0      0      0      0      0
## 14 64    40000      50      0      0      0      0      0      0
## 15 128   50000      50      0      0      0      0      0      0
## 16 128   40000      50      0      0      0      0      0      0
## 17 256   50000      50      0      0      0      0      0      0
## 18 256   40000      50      0      0      0      0      0      0

T 2

wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 2 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_uni_obj and filter(slices,
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 4

wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 4 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_uni_obj and filter(slices,
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 8

wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 8 & Generation == 40000)$pop_uni_obj,

```

```
alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_uni_obj and filter(slices, T == 8 & Ge
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 16 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_uni_obj and filter(slices, T == 16 &
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 32 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_uni_obj and filter(slices, T == 32 &
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 64
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 64 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_uni_obj and filter(slices, T == 64 &
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 128
```

```
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, T == 128 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$pop_uni_obj and filter(slices
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 256

wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, T == 256 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$pop_uni_obj and filter(slices
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

6.3.3.3 Activation gene coverage over time

```
lines = filter(tru_ot_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

## `summarise()` has grouped output by 'T'. You can override using the `groups`#
## argument.

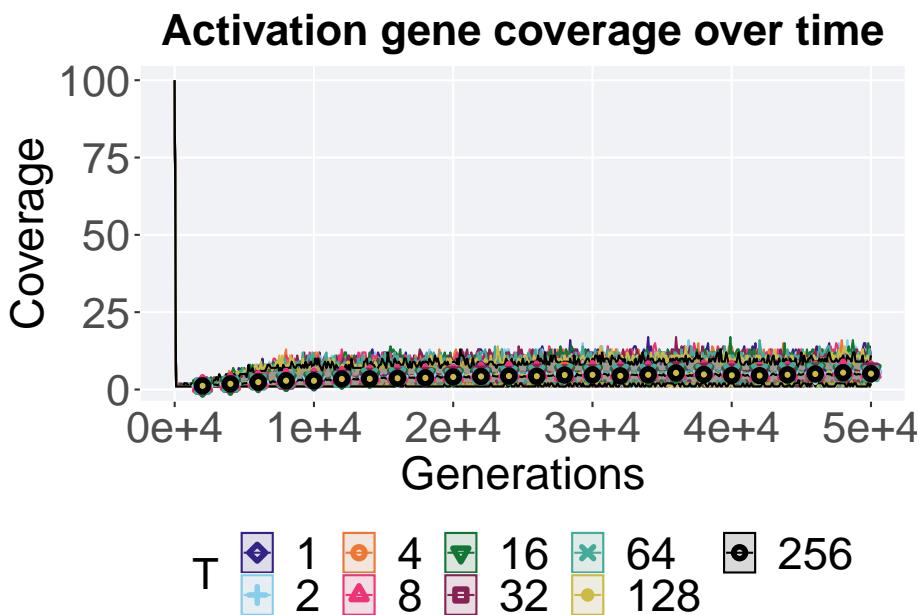
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2)
  scale_y_continuous(
    name="Coverage"
  ) +
  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

```



6.3.3.4 Activation gene coverage comparison

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

```

# 80% and final generation comparison
end = filter(tru_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 50000)
end$Generation <- factor(end$gen)

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

mvc_p = ggplot(mid, aes(x = T, y=uni_str_pos, group = T, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 100) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

```

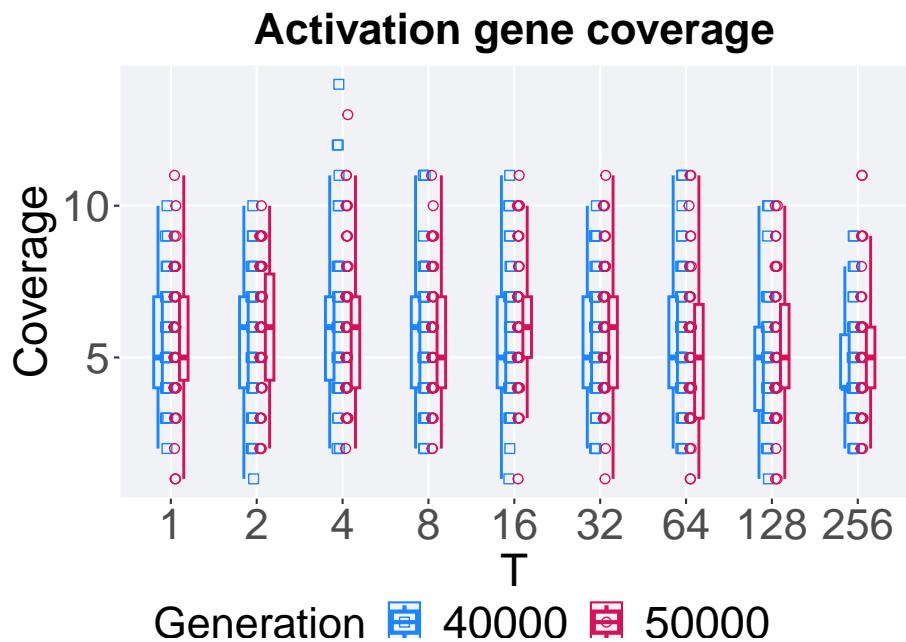
```

geom_point(data = end, aes(x = T, y=uni_str_pos), col = mvc_col[2], position = position_nudge(x = .1))
geom_boxplot(data = end, aes(x = T, y=uni_str_pos), position = position_nudge(x = .1))

scale_y_continuous(
  name="Coverage",
) +
scale_x_discrete(
  name="T"
) +
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") +
  theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



6.3.3.4.1 Stats

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

```
slices = filter(tru_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000, 40000))
slices %>%
  group_by(T, 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 'T'. You can override using the `.groups` argument.

## # A tibble: 18 x 9
## # Groups:   T [9]
##   T     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>      <int>   <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 1     50000       50     0     1     5   5.68    11   2.75
## 2 1     40000       50     0     2     5   5.54     10   3
## 3 2     50000       50     0     2     6   5.78     10   3.5
## 4 2     40000       50     0     1     6   5.64     10   3
## 5 4     50000       50     0     2     6   6.08     13   3
## 6 4     40000       50     0     2     6   6.12     14   2.75
## 7 8     50000       50     0     2     5   5.6      11   3
## 8 8     40000       50     0     2     6   5.8      11   3
## 9 16    50000       50     0     1     6   6.04     11   2
## 10 16   40000       50     0     1     5   5.42     11   3
## 11 32    50000       50     0     1     6   5.94     11   3
## 12 32   40000       50     0     2     5   5.58     10   3
## 13 64    50000       50     0     1     5   5.14     11   3.75
## 14 64   40000       50     0     2     5   5.66     11   3
## 15 128   50000       50     0     1     5   5.1      10   2.75
## 16 128   40000       50     0     1     5   4.78     10   2.75
## 17 256   50000       50     0     2     5   5.12     11   2
## 18 256   40000       50     0     2     4   4.58      9   1.75

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 2 & Generation == 40000)$uni_str_pos,
             alternative = 't')
```

```

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 2 & Generation == 50000)$uni_str_pos and filter(slices,  

## W = 1280.5, p-value = 0.8346  

## alternative hypothesis: true location shift is not equal to 0  

T 4  

wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, T == 4 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 4 & Generation == 50000)$uni_str_pos and filter(slices,  

## W = 1235, p-value = 0.9196  

## alternative hypothesis: true location shift is not equal to 0  

T 8  

wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, T == 8 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 8 & Generation == 50000)$uni_str_pos and filter(slices,  

## W = 1175, p-value = 0.6039  

## alternative hypothesis: true location shift is not equal to 0  

T 16  

wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, T == 16 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 16 & Generation == 50000)$uni_str_pos and filter(slices,  

## W = 1489.5, p-value = 0.09394  

## alternative hypothesis: true location shift is not equal to 0  

T 32  

wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, T == 32 & Generation == 40000)$uni_str_pos,

```

```
alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$uni_str_pos and filter(slices, T == 32 &
## W = 1363, p-value = 0.4333
## alternative hypothesis: true location shift is not equal to 0

T 64

wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 64 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$uni_str_pos and filter(slices, T == 64 &
## W = 1091, p-value = 0.2703
## alternative hypothesis: true location shift is not equal to 0

T 128

wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 128 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$uni_str_pos and filter(slices, T == 128 &
## W = 1335.5, p-value = 0.552
## alternative hypothesis: true location shift is not equal to 0

T 256

wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 256 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$uni_str_pos and filter(slices, T == 256 &
## W = 1439, p-value = 0.1846
## alternative hypothesis: true location shift is not equal to 0
```

6.4 Multi-path exploration results

Here we present the results for **best performances** and **activation gene coverage** found by each truncation selection size value replicate on the multi-path exploration diagnostic. Best performance found refers to the largest average trait score found in a given population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both values fall between 0 and 100.

6.4.1 Performance

Here we analyze the performances for each parameter replicate on the multi-path exploration diagnostic.

6.4.1.1 Performance over time

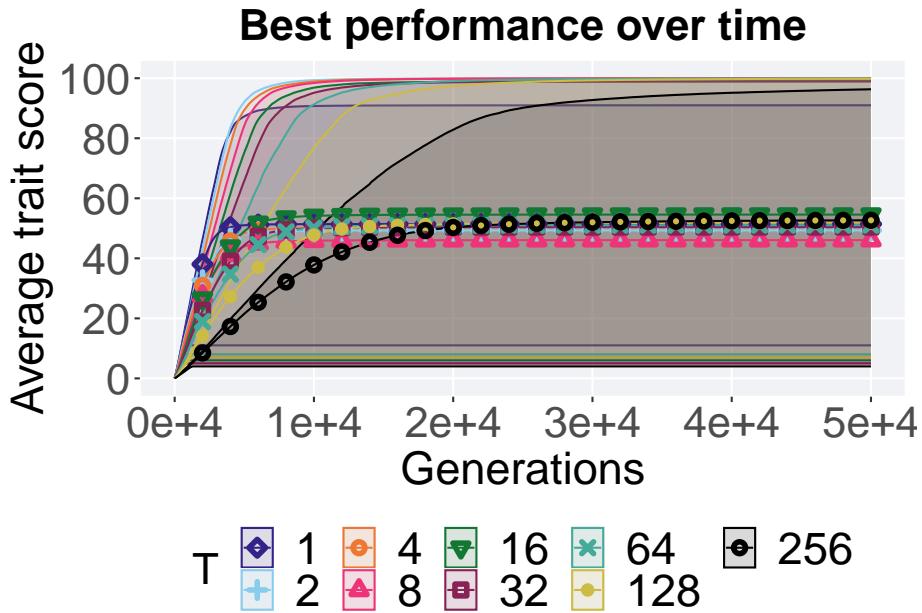
Performance over time.

```
lines = filter(tru_ot, diagnostic == 'multipath_exploration') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

## `summarise()` has grouped output by 'T'. You can override using the `.`groups` argument.

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = T, fill = T, shape = T,
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2),
  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(-1, 101),
    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("Best performance over time") +
  p_theme
```

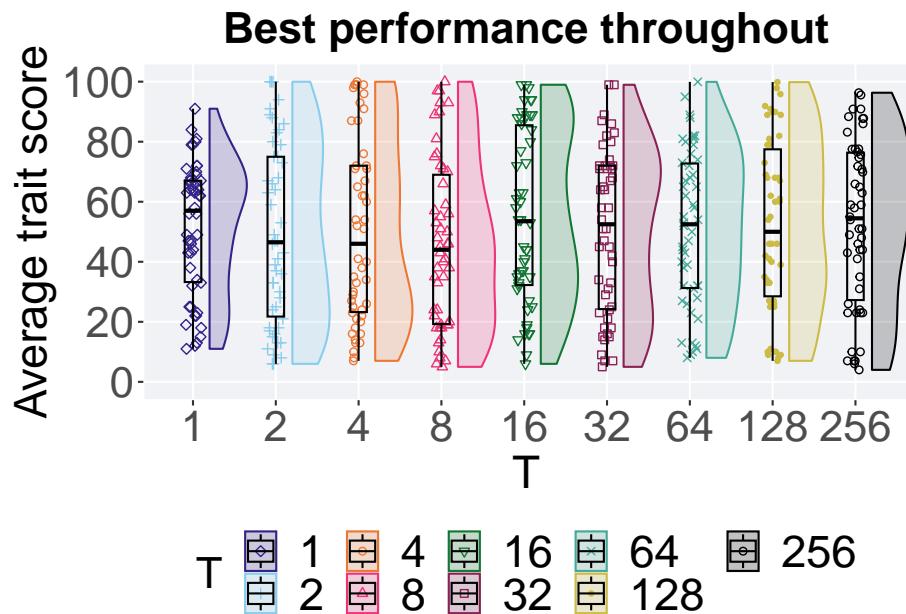


6.4.1.2 Best performance throughout

Here we plot the performance of the best performing solution found throughout 50,000 generations.

```
filter(tru_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = T, y = val / DIMENSIONALITY, color = T, fill = T, shape = T)) +
  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 = "T"
  ) +
  scale_shape_manual(values = SHAPE) +
```

```
scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best performance throughout") +
  p_theme
```



6.4.1.2.1 Stats

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

```
performance = filter(tru_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploratory') %>%
  group_by(performance, T) %>%
  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: 9 x 8
##   T      count na_cnt  min median  mean   max    IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1        50       0  11     57.0  51.4  91.0  33.7
```

```
## 2 2      50     0  6      46.5 49.3 100.   53.2
## 3 4      50     0  7.00   46.0 50.4 100.   48.7
## 4 8      50     0  5      44.0 46.1 100.   49.7
## 5 16     50     0  6      53.5 54.6 99.0   53.2
## 6 32     50     0  5      52.5 50.4 99.0   47.7
## 7 64     50     0  8.00   52.5 51.1 99.9   41.5
## 8 128    50     0  7      50.0 51.8 99.9   49.0
## 9 256    50     0  4      54.5 52.7 96.3   49.1
```

Kruskal–Wallis test provides evidence of no statistical differences among the best performing solution found throughout 50,000 generations.

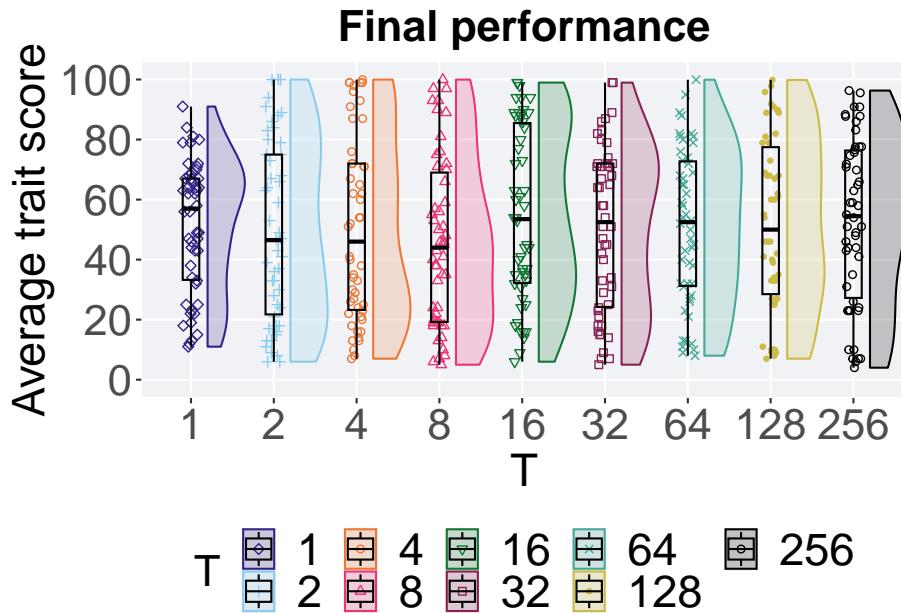
```
kruskal.test(val ~ T, data = performance)
```

```
##
## Kruskal-Wallis rank sum test
##
## data: val by T
## Kruskal-Wallis chi-squared = 2.7539, df = 8, p-value = 0.9488
```

6.4.1.3 End of 50,000 generations

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

```
filter(tru_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = T, y = pop_fit_max / DIMENSIONALITY, color = T, fill = T, shape = T)) +
  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 = "T"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final performance") +
  p_theme
```



6.4.1.3.1 Stats

Summary statistics for the best performance in the population at the end of 50,000 generations.

```
performance = filter(tru_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(performance, T) %>%
  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: 9 x 8
##   T      count na_cnt   min median   mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1       50     0 11     57.0  51.4  91.0  33.7
## 2 2       50     0  6     46.5  49.3 100.   53.2
## 3 4       50     0 7.00   46.0  50.4 100.   48.7
## 4 8       50     0  5     44.0  46.1 100.   49.7
## 5 16      50     0  6     53.5  54.6  99.0  53.2
## 6 32      50     0  5     52.5  50.4  99.0  47.7
```

```
## 7 64      50      0  8.00   52.5  51.1  99.9  41.5
## 8 128     50      0  7      50.0  51.8  99.9  49.0
## 9 256     50      0  4      54.5  52.7  96.3  49.1
```

Kruskal–Wallis test provides evidence of no statistical differences among best performance in the population at the end of 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: pop_fit_max by T
## Kruskal-Wallis chi-squared = 2.7539, df = 8, p-value = 0.9488
```

6.4.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the multi-path exploration diagnostic.

6.4.2.1 Coverage over time

Activation gene coverage over time.

```
lines = filter(tru_ot, diagnostic == 'multipath_exploration') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

## `summarise()` has grouped output by 'T'. You can override using the `.`groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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(-1, 101),
    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)+  

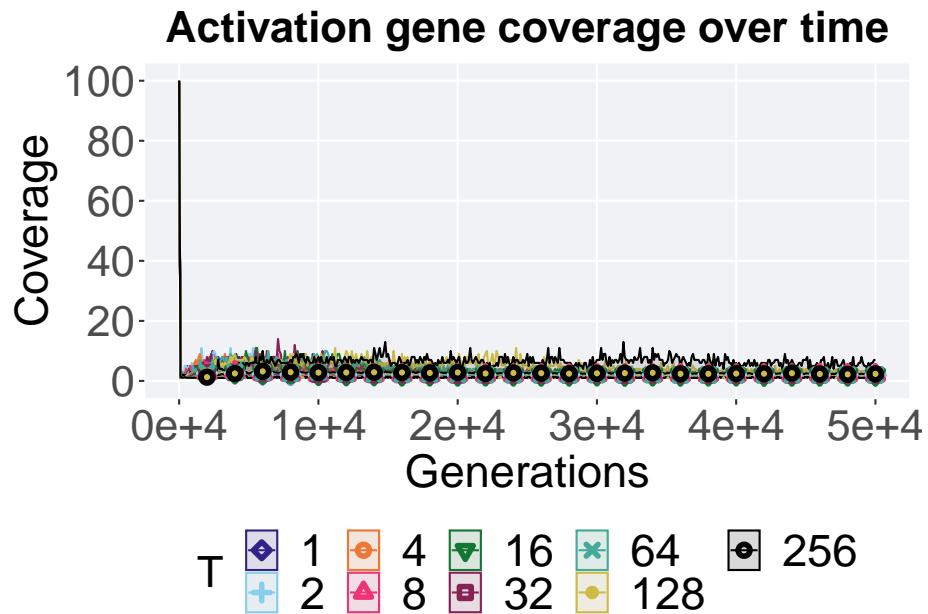
scale_colour_manual(values = cb_palette) +  

scale_fill_manual(values = cb_palette) +  

ggtitle("Activation gene coverage over time") +  

p_theme

```



6.4.2.2 End of 50,000 generations

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

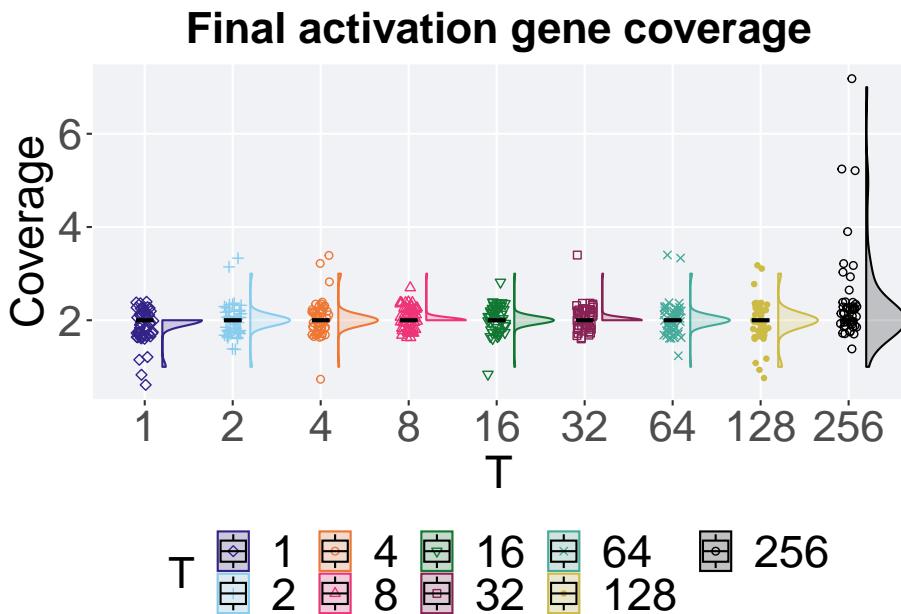
```

filter(tru_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
ggplot(., aes(x = T, y = uni_str_pos, color = T, fill = T, shape = T)) +
  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"
  ) +
  scale_x_discrete(
    name="T"
  )+
  scale_shape_manual(values=SHAPE)+
  scale_fill_manual(values=cb_palette)+  

  scale_color_manual(values=cb_palette)

```

```
scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtile("Final activation gene coverage") +
  p_theme
```



6.4.2.2.1 Stats

Summary statistics for the activation gene coverage in the population at the end of 50,000 generations.

```
coverage = filter(tru_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(coverage, T) %>%
  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: 9 x 8
##   T      count  na_cnt  min  median  mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 1        50       0     0     1     2  1.92     2
## 2 2        50       0     0     1     2  1.92     2
## 3 4        50       0     0     1     2  1.92     2
## 4 8        50       0     0     1     2  1.92     2
## 5 16       50       0     0     1     2  1.92     2
## 6 32       50       0     0     1     2  1.92     2
## 7 64       50       0     0     1     2  1.92     2
## 8 128      50       0     0     1     2  1.92     2
## 9 256      50       0     0     1     2  1.92     2
```

```
## 2 2      50     0     1     2 2     3     0
## 3 4      50     0     1     2 2.04   3     0
## 4 8      50     0     2     2 2.02   3     0
## 5 16     50     0     1     2 2     3     0
## 6 32     50     0     2     2 2.02   3     0
## 7 64     50     0     1     2 2.02   3     0
## 8 128    50     0     1     2 1.98   3     0
## 9 256    50     0     1     2 2.34   7     0
```

Kruskal–Wallis test provides evidence of statistical differences among activation gene coverage in the population at the end of 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: uni_str_pos by T
## Kruskal-Wallis chi-squared = 20.807, df = 8, p-value = 0.007679
```

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

```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$T, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$uni_str_pos and coverage$T
##
##      1     2     4     8    16    32    64   128
## 2 1.000 -     -     -     -     -     -     -
## 4 1.000 1.000 -     -     -     -     -     -
## 8 0.911 1.000 1.000 -     -     -     -     -
## 16 1.000 1.000 1.000 1.000 -     -     -     -
## 32 0.911 1.000 1.000 1.000 1.000 -     -     -
## 64 1.000 1.000 1.000 1.000 1.000 1.000 -     -
## 128 1.000 1.000 1.000 1.000 1.000 1.000 1.000 -
## 256 0.047 0.915 1.000 0.887 0.569 0.887 1.000 0.866
##
## P value adjustment method: bonferroni
```

6.4.3 Multi-valley crossing

6.4.3.1 Performance over time

```

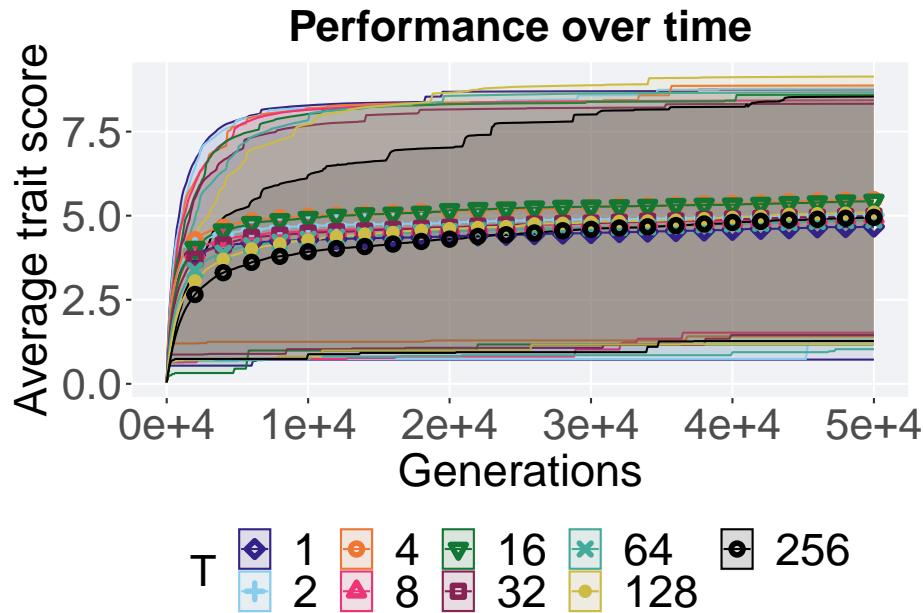
# data for lines and shading on plots
lines = filter(tru_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
    mean = mean(pop_fit_max) / DIMENSIONALITY,
    max = max(pop_fit_max) / DIMENSIONALITY
  )

## `summarise()` has grouped output by 'T'. You can override using the `.`groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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="Average trait score"
  ) +
  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 +
  guides(
    shape=guide_legend(nrow=2, title.position = "left"),
    color=guide_legend(nrow=2, title.position = "left"),
    fill=guide_legend(nrow=2, title.position = "left")
  )

```



6.4.3.2 Performance comparison

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

```
# 80% and final generation comparison
end = filter(tru_ot_mvc, diagnostic == 'multipath_exploration' & gen == 50000 & T != '1')
end$Generation <- factor(end$gen)

mid = filter(tru_ot_mvc, diagnostic == 'multipath_exploration' & gen == 40000 & T != '1')
mid$Generation <- factor(mid$gen)

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

  geom_point(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2])
  geom_boxplot(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 1), lwd = 0.7, col = mvc_col[2])

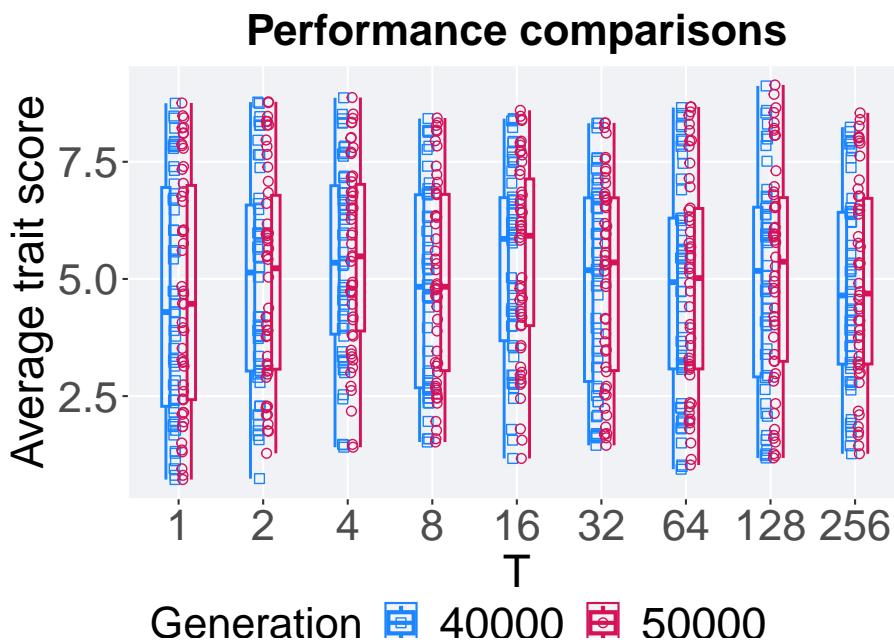
  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="T"
  ) +
  scale_shape_manual(values=c(0,1)) +
```

```

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
)

```



6.4.3.3 Stats

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

```

slices = filter(tru_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max / DIMENSIONALITY)),
    mean = mean(pop_fit_max / DIMENSIONALITY),
    median = median(pop_fit_max / DIMENSIONALITY),
    min = min(pop_fit_max / DIMENSIONALITY),
    max = max(pop_fit_max / DIMENSIONALITY),
    stdev = sd(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 'T'. You can override using the `groups`#
## argument.

## # A tibble: 18 x 9
## # Groups:   T [9]
##   T     Generation count na_cnt   min median   mean   max   IQR
##   <fct>    <fct>     <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##   1 1      50000       50     0 0.720    4.47    4.68    8.75   4.57
##   2 1      40000       50     0 0.720    4.29    4.58    8.75   4.67
##   3 2      50000       50     0 1.28     5.23    5.02    8.78   3.71
##   4 2      40000       50     0 0.740    5.14    4.94    8.77   3.54
##   5 4      50000       50     0 1.41     5.48    5.48    8.87   3.13
##   6 4      40000       50     0 1.41     5.35    5.37    8.87   3.17
##   7 8      50000       50     0 1.52     4.83    4.96    8.43   3.76
##   8 8      40000       50     0 1.52     4.83    4.85    8.42   4.12
##   9 16     50000       50     0 1.17     5.92    5.44    8.60   3.13
##  10 16    40000       50     0 1.17     5.85    5.33    8.42   3.05
##  11 32     50000       50     0 1.45     5.35    4.99    8.33   3.69
##  12 32     40000       50     0 1.45     5.19    4.90    8.32   3.92
##  13 64     50000       50     0 1.03     5.02    4.87    8.67   3.42
##  14 64     40000       50     0 0.940    4.93    4.75    8.66   3.22
##  15 128    50000       50     0 1.18     5.37    5.10    9.14   3.49
##  16 128    40000       50     0 1.18     5.17    4.95    9.12   3.62
##  17 256    50000       50     0 1.27     4.69    4.95    8.54   3.53
##  18 256    40000       50     0 1.27     4.65    4.80    8.24   3.24

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1300, p-value = 0.7329
## alternative hypothesis: true location shift is not equal to 0

T 4

```

```
wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 4 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_fit_max and filter(slices, T == 4 & Ge
## W = 1307.5, p-value = 0.6944
## alternative hypothesis: true location shift is not equal to 0

T 8

wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 8 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_fit_max and filter(slices, T == 8 & Ge
## W = 1317, p-value = 0.6466
## alternative hypothesis: true location shift is not equal to 0

T 16

wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 16 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_fit_max and filter(slices, T == 16 &
## W = 1320, p-value = 0.6319
## alternative hypothesis: true location shift is not equal to 0

T 32

wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 32 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_fit_max and filter(slices, T == 32 &
## W = 1298.5, p-value = 0.7407
## alternative hypothesis: true location shift is not equal to 0
```

T 64

```
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 64 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1306, p-value = 0.702
## alternative hypothesis: true location shift is not equal to 0
```

T 128

```
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 128 & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1315.5, p-value = 0.6541
## alternative hypothesis: true location shift is not equal to 0
```

T 256

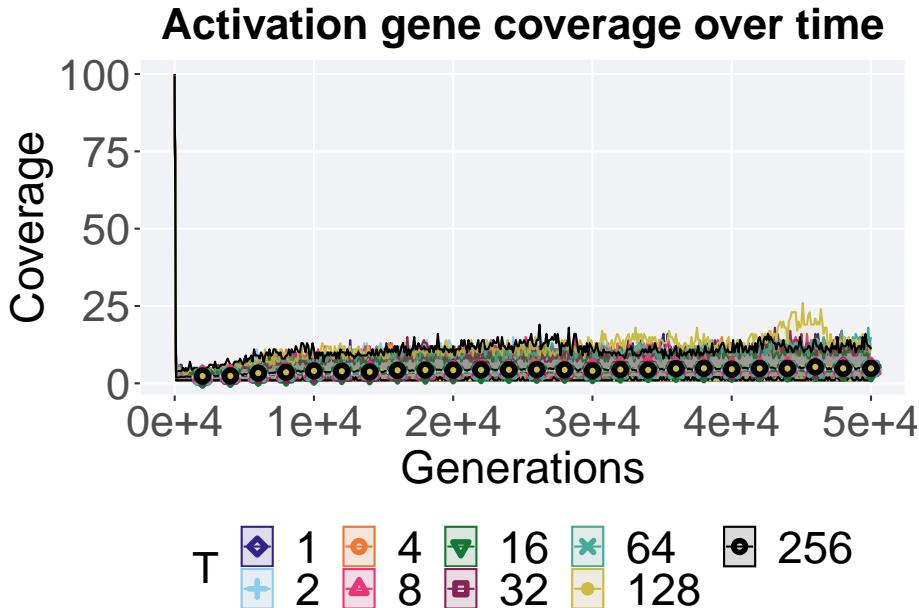
```
wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 256 & Generation == 40000)$pop_fit_max,
            alternative = 't')
```

```
##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1328.5, p-value = 0.5908
## alternative hypothesis: true location shift is not equal to 0
```

6.4.3.4 Activation gene coverage over time

```
lines = filter(tru_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )
```

```
## `summarise()` has grouped output by 'T'. You can override using the `groups`  
## argument.  
  
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +  
  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 =  
    1) +  
  scale_y_continuous(  
    name="Coverage"  
) +  
  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
```



6.4.3.5 Activation gene coverage comparison

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

```

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

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

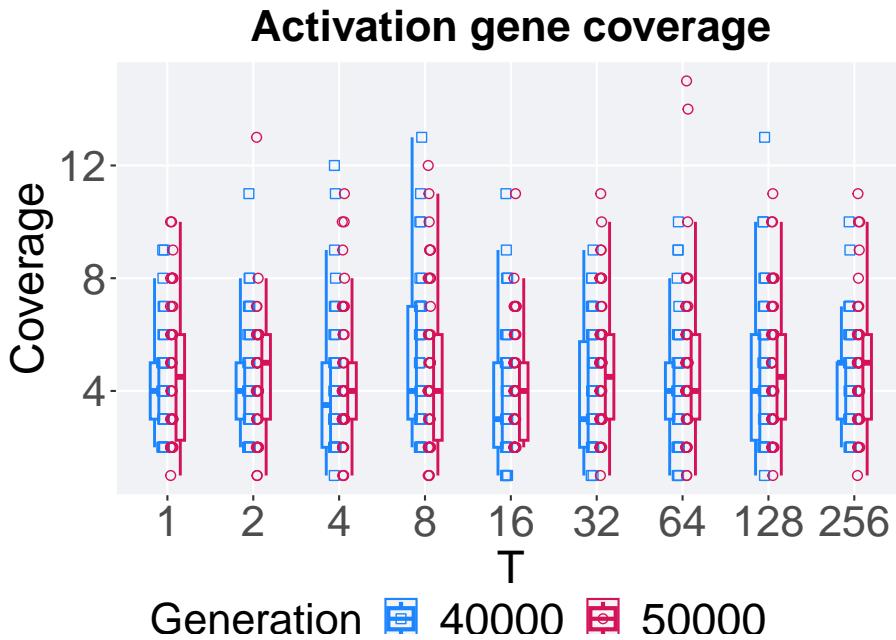
mvc_p = ggplot(mid, aes(x = T, y=uni_str_pos, group = T, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), lwd = 1, col = mvc_col[1])
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

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

  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="T"
  )+
  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") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



6.4.3.5.1 Stats

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

```
slices = filter(tru_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, 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 'T'. You can override using the `.groups` argument.

## # A tibble: 18 x 9
## # Groups:   T [9]
##   T     Generation count na_cnt  min median  mean   max   IQR
##   <fct> <fct>      <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
```

```

##  1 1      50000      50      0      1     4.5   4.68    10   3.75
##  2 1      40000      50      0      2     4     4.12     9   2
##  3 2      50000      50      0      1     5     4.5    13   3
##  4 2      40000      50      0      2     4     4.24    11   2
##  5 4      50000      50      0      1     4     4.3    11   2
##  6 4      40000      50      0      1     3.5   4.06    12   3
##  7 8      50000      50      0      1     4     4.8    12   3.75
##  8 8      40000      50      0      2     4     4.78    13   4
##  9 16     50000      50      0      2     4     4.12    11  2.75
## 10 16     40000      50      0      1     3     3.96    11   3
## 11 32     50000      50      0      1     4.5   4.78    11   3
## 12 32     40000      50      0      1     3     3.96     9   3.75
## 13 64     50000      50      0      1     4     4.82    15   3
## 14 64     40000      50      0      1     4     4.12    10   2
## 15 128    50000      50      0      1     4.5   4.78    11   3
## 16 128    40000      50      0      1     4     4.52    13  3.75
## 17 256    50000      50      0      1     5     4.82    11   3
## 18 256    40000      50      0      2     5     4.56    10   2

T 2

wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 2 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1373, p-value = 0.3915
## alternative hypothesis: true location shift is not equal to 0

T 4

wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 4 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1354, p-value = 0.4688
## alternative hypothesis: true location shift is not equal to 0

T 8

wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 8 & Generation == 40000)$uni_str_pos,

```

```

    alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$uni_str_pos and filter(slices, T == 8 & Ge
## W = 1254.5, p-value = 0.9778
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 16 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$uni_str_pos and filter(slices, T == 16 &
## W = 1327, p-value = 0.5923
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 32 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$uni_str_pos and filter(slices, T == 32 &
## W = 1510.5, p-value = 0.06951
## alternative hypothesis: true location shift is not equal to 0

T 64
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 64 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$uni_str_pos and filter(slices, T == 64 &
## W = 1407, p-value = 0.2749
## alternative hypothesis: true location shift is not equal to 0

T 128

```

```
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 128 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$uni_str_pos and filter(slices
## W = 1348.5, p-value = 0.4941
## alternative hypothesis: true location shift is not equal to 0
T 256

wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$uni_str_pos,
            y = filter(slices, T == 256 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$uni_str_pos and filter(slices
## W = 1308, p-value = 0.6875
## alternative hypothesis: true location shift is not equal to 0
```

Chapter 7

Tournament selection

We present the results from our parameter sweep on tournament selection. 50 replicates are conducted for each tournament size T parameter value explored.

```
library(ggplot2)
library(cowplot)
library(dplyr)
library(PupilometryR)
```

7.1 Exploitation rate results

Here we present the results for **best performances** found by each tournament selection value replicate on the exploitation rate diagnostic.

7.1.1 Performance over time

Performance over time.

```
lines = filter(tor_ot, diagnostic == 'exploitation_rate') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

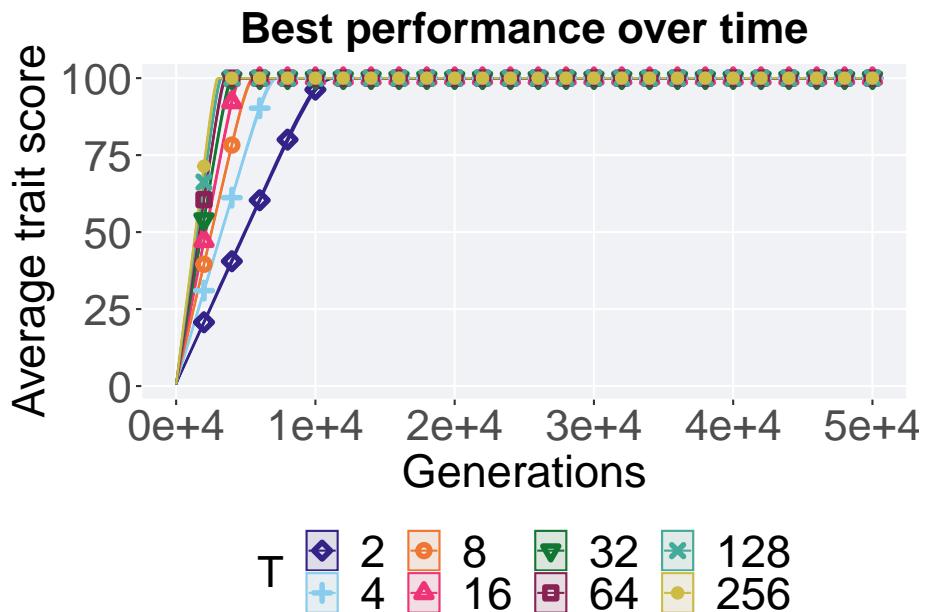
## `summarise()` has grouped output by 'T'. You can override using the `.`groups` argument.

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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"
) +
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("Best performance over time") +
p_theme

```



7.1.2 Generation satisfactory solution found

The first Generations a satisfactory solution is found throughout the 50,000 generations.

```

filter(tor_ssf, Diagnostic == 'EXPLOITATION_RATE') %>%
ggplot(., aes(x = T, y = Generations, color = T, fill = T, shape = T)) +

```

```

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_shape_manual(values=SHAPE) +
  scale_y_continuous(
    name="Generation",
    limits=c(0, 12000),
    breaks=c(0, 2000, 4000, 6000, 8000, 10000, 12000),
    labels=c("0e+4", "2e+4", "4e+4", "6e+4", "8e+4", "10e+4", "12e+4")
  ) +
  scale_x_discrete(
    name="T"
  ) +
  ggtitle("Generation satisfactory solution found") +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  p_theme

```



7.1.2.1 Stats

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

```

ssf = filter(tor_ssf, Diagnostic == 'EXPLOITATION_RATE')
group_by(ssf, T) %>%

```

```
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: 8 x 8
##   T      count na_cnt   min median   mean   max   IQR
##   <fct> <int>    <int> <dbl> <dbl> <int> <dbl>
## 1 2        50       0 10815 10955 10966. 11157 102.
## 2 4        50       0  6960  7059  7052.  7113  48.8
## 3 8        50       0  5403  5457  5453.  5519  51.8
## 4 16       50       0  4471  4530. 4532.  4597  33.8
## 5 32       50       0  3865  3938  3939.  4009  31.8
## 6 64       50       0  3446  3504. 3502.  3556  24.2
## 7 128      50       0  3152  3192  3190.  3228  26.8
## 8 256      50       0  2912  2946. 2950.  2985  24.8
```

Kruskal–Wallis test provides evidence of significant differences among the Generations a satisfactory solution is first found.

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

##
## Kruskal-Wallis rank sum test
##
## data: Generations by T
## 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 on the Generations a satisfactory solution is first found.

pairwise.wilcox.test(x = ssf$Generations, g = ssf$T, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 'l')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: ssf$Generations and ssf$T
##
##      2      4      8     16     32     64    128
## 4 <2e-16 -      -      -      -      -      -
## 8 <2e-16 <2e-16 -      -      -      -      -
## 16 <2e-16 <2e-16 <2e-16 -      -      -      -
```

```
## 32 <2e-16 <2e-16 <2e-16 <2e-16 - - -
## 64 <2e-16 <2e-16 <2e-16 <2e-16 - - -
## 128 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 - -
## 256 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

7.1.3 Multi-valley crossing

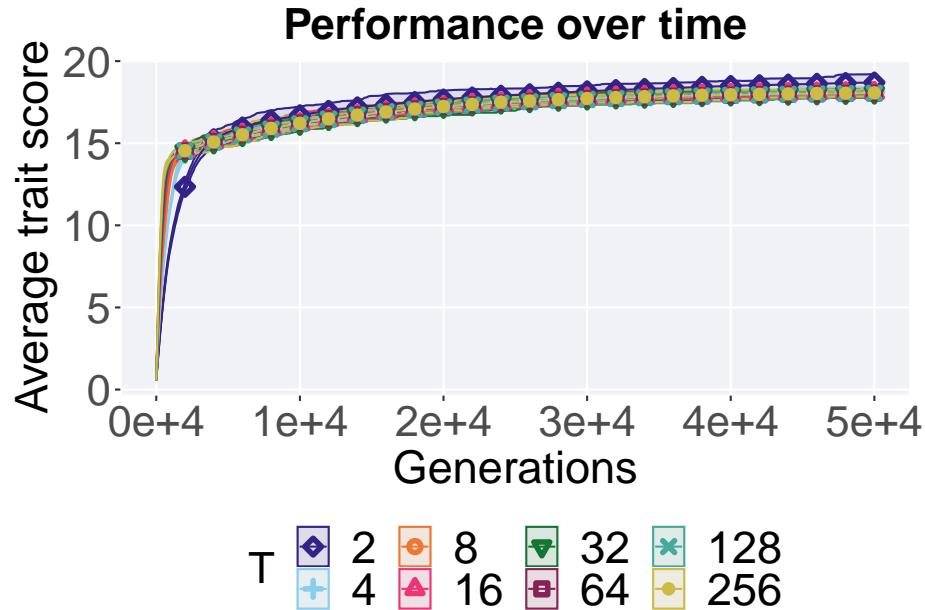
7.1.3.1 Performance over time

```
# data for lines and shading on plots
lines = filter(tor_ot_mvc, diagnostic == 'exploitation_rate') %>%
  group_by(T, 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 'T'. You can override using the `.`groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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="Average trait score"
  ) +
  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 +
  guides(
    shape=guide_legend(nrow=2, title.position = "left"),
    color=guide_legend(nrow=2, title.position = "left"),
    fill=guide_legend(nrow=2, title.position = "left")
  )
```



7.1.3.2 Performance comparison

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

```
# 80% and final generation comparison
end = filter(tor_ot_mvc, diagnostic == 'exploitation_rate' & gen == 50000 & T != 'ran')
end$Generation <- factor(end$gen)

mid = filter(tor_ot_mvc, diagnostic == 'exploitation_rate' & gen == 40000 & T != 'ran')
mid$Generation <- factor(mid$gen)

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

  geom_point(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], na.rm = TRUE)
  geom_boxplot(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 1), lwd = 0.7, col = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="T"
  ) +
```

```

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,  

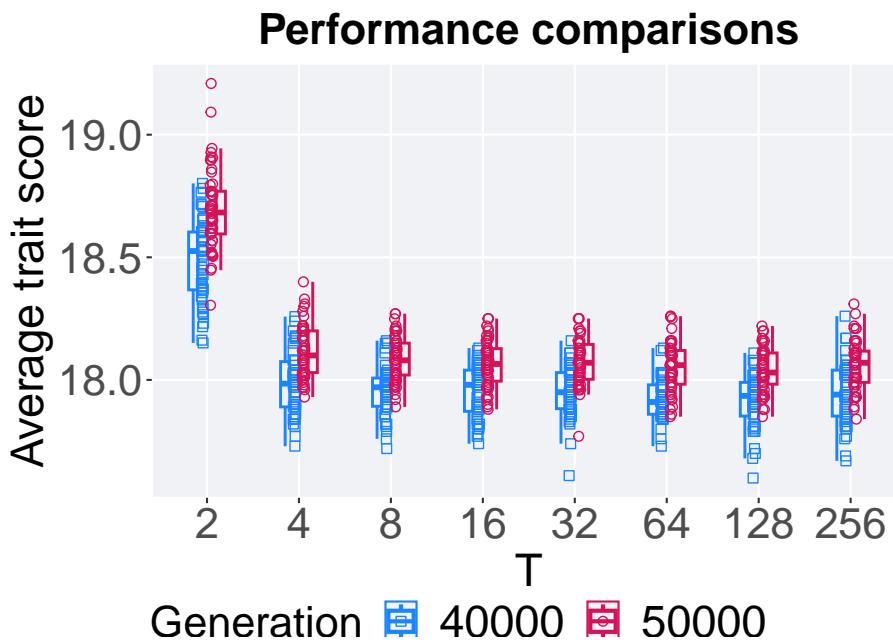
  nrow=2,  

  rel_heights = c(1,.05),  

  label_size = TSIZE  

)

```



7.1.3.3 Stats

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

```

slices = filter(tor_ot_mvc, diagnostic == 'exploitation_rate' & (gen == 50000 | gen == 40000))  

slices$Generation <- factor(slices$gen, levels = c(50000,40000))  

slices %>%  

  group_by(T, 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 'T'. You can override using the `groups`#
## argument.

## # A tibble: 16 x 9
## # Groups:   T [8]
##   T     Generation count na_cnt    min median   mean   max   IQR
##   <fct> <fct>      <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2      50000       50    0  18.3  18.7  18.7  19.2 0.174
## 2 2      40000       50    0  18.2  18.5  18.5  18.8 0.236
## 3 4      50000       50    0  17.9  18.1  18.1  18.4 0.170
## 4 4      40000       50    0  17.7  18.0  18.0  18.3 0.185
## 5 8      50000       50    0  17.9  18.1  18.1  18.3 0.130
## 6 8      40000       50    0  17.7  18.0  18.0  18.2 0.115
## 7 16     50000       50    0  17.9  18.1  18.1  18.2 0.133
## 8 16     40000       50    0  17.7  18.0  18.0  18.1 0.168
## 9 32     50000       50    0  17.8  18.1  18.1  18.2 0.142
## 10 32    40000       50    0  17.6  17.9  18.0  18.2 0.147
## 11 64     50000       50    0  17.8  18.1  18.1  18.3 0.137
## 12 64     40000       50    0  17.7  17.9  17.9  18.1 0.120
## 13 128    50000       50    0  17.8  18.0  18.0  18.2 0.128
## 14 128    40000       50    0  17.6  17.9  17.9  18.1 0.137
## 15 256    50000       50    0  17.8  18.1  18.1  18.3 0.128
## 16 256    40000       50    0  17.7  17.9  18.0  18.3 0.188

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 2007, p-value = 1.836e-07
## alternative hypothesis: true location shift is not equal to 0

T 4

```

```
wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 4 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_fit_max and filter(slices, T == 4 & Ge
## W = 1961, p-value = 9.59e-07
## alternative hypothesis: true location shift is not equal to 0

T 8
wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 8 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_fit_max and filter(slices, T == 8 & Ge
## W = 2075, p-value = 1.301e-08
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 16 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_fit_max and filter(slices, T == 16 &
## W = 1948.5, p-value = 1.483e-06
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 32 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_fit_max and filter(slices, T == 32 &
## W = 2020.5, p-value = 1.097e-07
## alternative hypothesis: true location shift is not equal to 0
```

T 64

```
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 64 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 2104.5, p-value = 3.84e-09
## alternative hypothesis: true location shift is not equal to 0
```

T 128

```
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 128 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 2003.5, p-value = 2.068e-07
## alternative hypothesis: true location shift is not equal to 0
```

T 256

```
wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 256 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1892.5, p-value = 9.536e-06
## alternative hypothesis: true location shift is not equal to 0
```

7.2 Ordered exploitation results

Here we present the results for **best performances** found by each tournament selection size value 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 and 100.

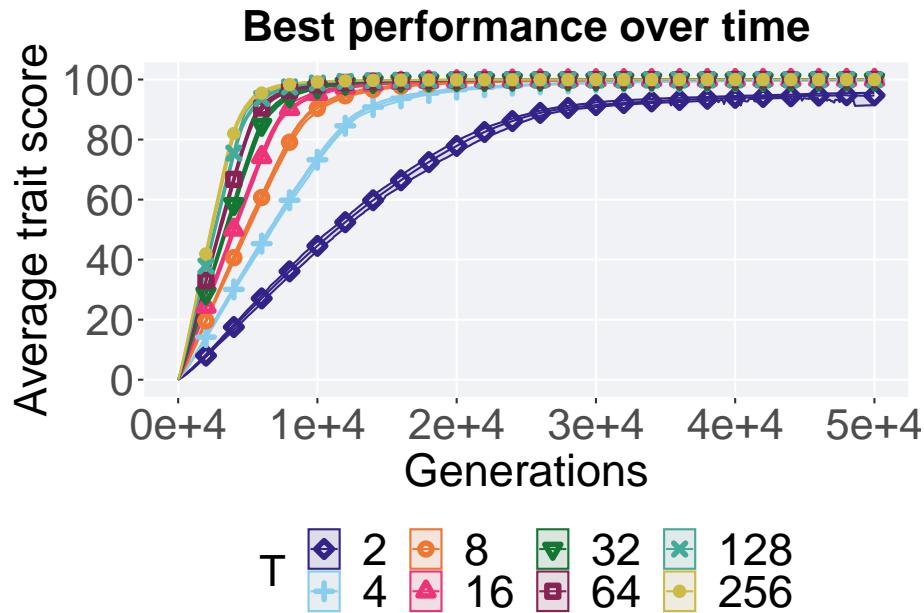
7.2.1 Performance over time

Performance over time.

```
lines = filter(tor_ot, diagnostic == 'ordered_exploitation') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

## `summarise()` has grouped output by 'T'. You can override using the ` `.groups` ` argument.

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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="Average trait score",
    limits=c(-1, 101),
    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("Best performance over time") +
  p_theme
```

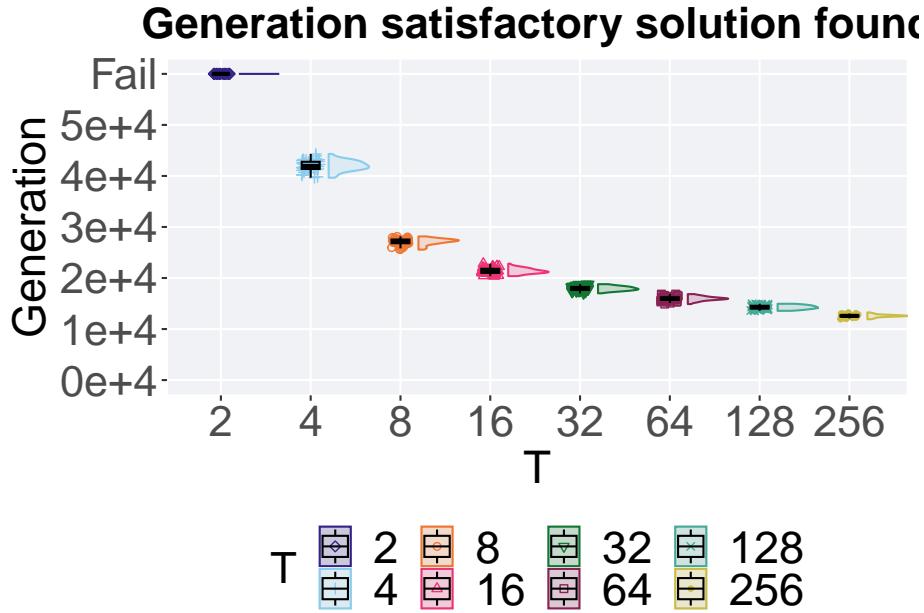


7.2.2 Generation satisfactory solution found

The first Generations a satisfactory solution is found throughout the 50,000 generations.

```
filter(tor_ssf, Diagnostic == 'ORDERED_EXPLOITATION') %>%
  ggplot(., aes(x = T, y = Generations, color = T, fill = T, shape = T)) +
  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_shape_manual(values=SHAPE) +
  scale_y_continuous(
    name="Generation",
    limits=c(0, 60000),
    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="T"
  ) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Generation satisfactory solution found") +
  p_theme
```

```
## Warning: Removed 26 rows containing missing values (`geom_point()`).
```



7.2.2.1 Stats

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

```
ssf = filter(tor_ssf, Diagnostic == 'ORDERED_EXPLOITATION')
group_by(ssf, T) %>%
  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: 8 x 8
##   T      count na_cnt  min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <int> <dbl>
## 1 2        50     0 60000 60000  60000  60000  0
## 2 4        50     0 39659 41972. 41991. 44310 1404.
## 3 8        50     0 25563 27254. 27122. 28151  714
## 4 16       50     0 20374 21270. 21366. 22808  792.
```

```
## 5 32      50      0 17005 17945 17950. 18792 572.
## 6 64      50      0 14896 15960 15965. 16845 480.
## 7 128     50      0 13569 14278. 14278. 14952 545
## 8 256     50      0 11919 12578. 12580. 13249 262.
```

Kruskal–Wallis test provides evidence of significant differences among the first Generations a satisfactory solution is found throughout the 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: Generations by T
## Kruskal-Wallis chi-squared = 393.52, df = 7, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the first Generations a satisfactory solution is found throughout the 50,000 generations.

```
pairwise.wilcox.test(x = ssf$Generations, g = ssf$T , p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 'l')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: ssf$Generations and ssf$T
##
##      2      4      8      16      32      64     128
## 4 <2e-16 -      -      -      -      -      -
## 8 <2e-16 <2e-16 -      -      -      -      -
## 16 <2e-16 <2e-16 <2e-16 -      -      -      -
## 32 <2e-16 <2e-16 <2e-16 <2e-16 -      -      -
## 64 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -      -
## 128 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 -
## 256 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16
##
## P value adjustment method: bonferroni
```

7.2.3 Multi-valley crossing

7.2.3.1 Performance over time

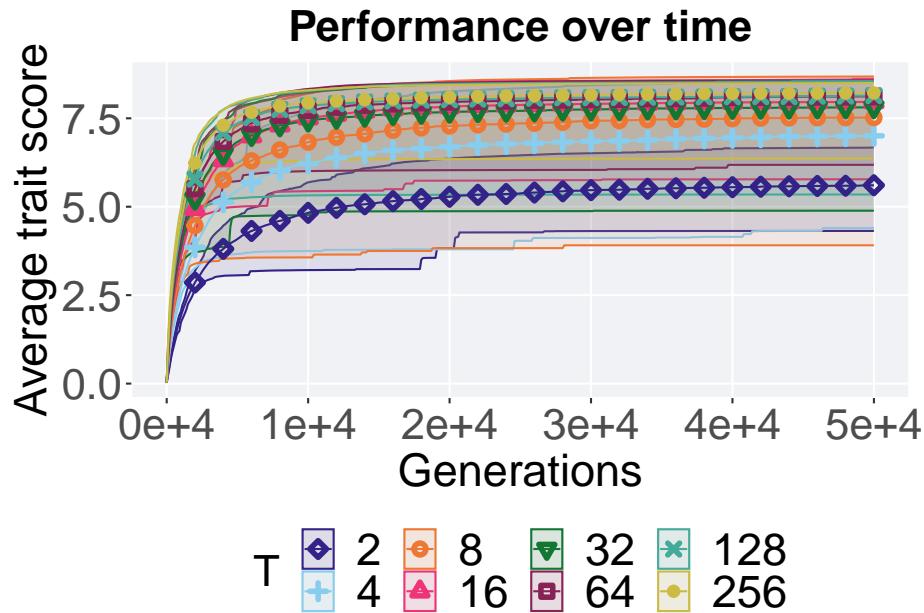
```
# data for lines and shading on plots
lines = filter(tor_ot_mvc, diagnostic == 'ordered_exploitation') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max) / DIMENSIONALITY,
```

```
mean = mean(pop_fit_max) / DIMENSIONALITY,
max = max(pop_fit_max) / DIMENSIONALITY
)
```

```
## `summarise()` has grouped output by 'T'. You can override using the `groups`#
## argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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="Average trait score"
  ) +
  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 +
  guides(
    shape=guide_legend(nrow=2, title.position = "left"),
    color=guide_legend(nrow=2, title.position = "left"),
    fill=guide_legend(nrow=2, title.position = "left")
  )
)
```



7.2.3.2 Performance comparison

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

```
# 80% and final generation comparison
end = filter(tor_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 50000 & T != 'random')
end$Generation <- factor(end$gen)

mid = filter(tor_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 40000 & T != 'random')
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = T, y=pop_fit_max / DIMENSIONALITY, group = T, shape = Generation))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), size = 10)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

  geom_point(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], size = 10)
  geom_boxplot(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 1), lwd = 0.7, col = mvc_col[2])

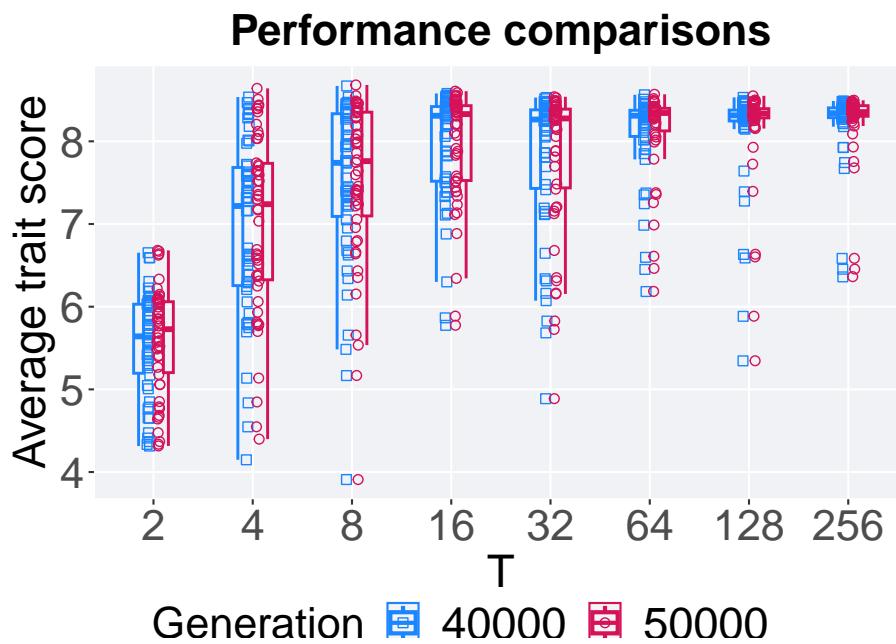
  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="T"
  ) +
  scale_shape_manual(values=c(0,1)) +
```

```

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
)

```



7.2.3.3 Stats

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

```

slices = filter(tor_ot_mvc, diagnostic == 'ordered_exploitation' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, 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 'T'. You can override using the `groups`#
## argument.

## # A tibble: 16 x 9
## # Groups:   T [8]
##   T     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>      <int> <int> <dbl> <dbl> <dbl> <dbl>
## 1 2     50000       50    0  4.32  5.73  5.61  6.68 0.858
## 2 2     40000       50    0  4.31  5.64  5.55  6.65 0.835
## 3 4     50000       50    0  4.40  7.24  7.00  8.64 1.41
## 4 4     40000       50    0  4.15  7.22  6.95  8.54 1.43
## 5 8     50000       50    0  3.91  7.76  7.52  8.68 1.26
## 6 8     40000       50    0  3.91  7.74  7.49  8.67 1.24
## 7 16    50000       50    0  5.78  8.33  7.96  8.61 0.905
## 8 16    40000       50    0  5.77  8.31  7.95  8.58 0.903
## 9 32    50000       50    0  4.89  8.28  7.81  8.54 0.952
## 10 32   40000       50    0  4.89  8.26  7.79  8.53 0.950
## 11 64   50000       50    0  6.18  8.34  8.12  8.57 0.273
## 12 64   40000       50    0  6.18  8.31  8.09  8.56 0.316
## 13 128  50000       50    0  5.35  8.34  8.14  8.55 0.112
## 14 128  40000       50    0  5.34  8.31  8.11  8.53 0.132
## 15 256  50000       50    0  6.36  8.35  8.22  8.50 0.128
## 16 256  40000       50    0  6.36  8.34  8.20  8.49 0.119

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_fit_max and filter(slices, T == 2 & Generation == 40000)$pop_fit_max
## W = 1337, p-value = 0.551
## alternative hypothesis: true location shift is not equal to 0

T 4
wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 4 & Generation == 40000)$pop_fit_max,
```

```
alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_fit_max and filter(slices, T == 4 & Ge
## W = 1315, p-value = 0.6566
## alternative hypothesis: true location shift is not equal to 0

T 8
wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 8 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_fit_max and filter(slices, T == 8 & Ge
## W = 1306.5, p-value = 0.6995
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 16 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_fit_max and filter(slices, T == 16 &
## W = 1309, p-value = 0.6867
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 32 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_fit_max and filter(slices, T == 32 &
## W = 1305, p-value = 0.7071
## alternative hypothesis: true location shift is not equal to 0

T 64
```

```
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 64 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1379, p-value = 0.3757
## alternative hypothesis: true location shift is not equal to 0

T 128

wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 128 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1409, p-value = 0.2745
## alternative hypothesis: true location shift is not equal to 0

T 256

wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 256 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1381, p-value = 0.3683
## alternative hypothesis: true location shift is not equal to 0
```

7.3 Contraditory objectives diagnostic

Here we present the results for **satisfactory trait coverage** and **activation gene coverage** found by each tournament selection size value replicate on the ordered exploitation diagnostic. Satisfactory trait coverage refers to the count of unique satisfied traits in the population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both coverage values fall between 0 and 100.

7.3.1 Satisfactory trait coverage

Satisfactory trait coverage analysis.

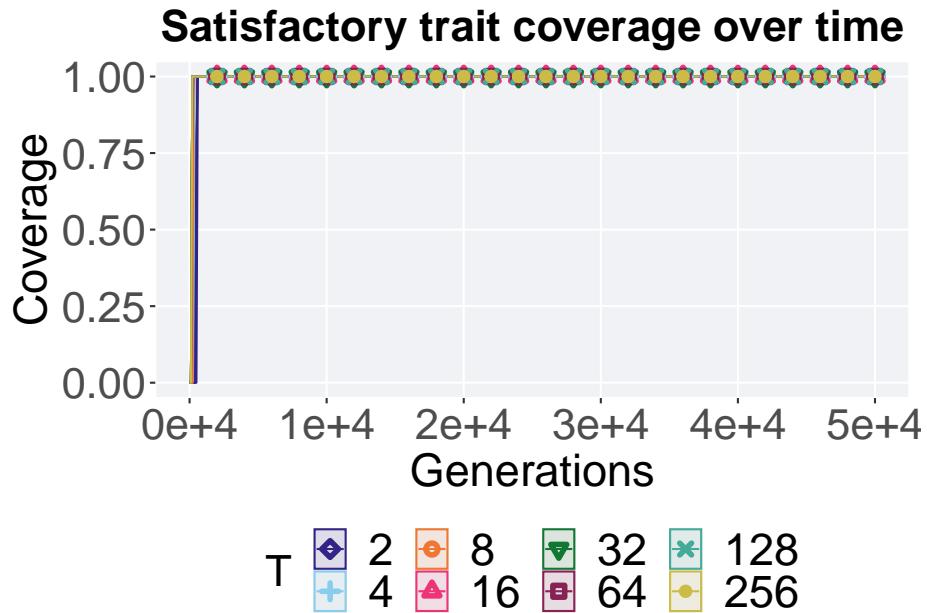
7.3.1.1 Coverage over time

Satisfactory trait coverage over time.

```
lines = filter(tor_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

## `summarise()` has grouped output by 'T'. You can override using the `groups` argument.

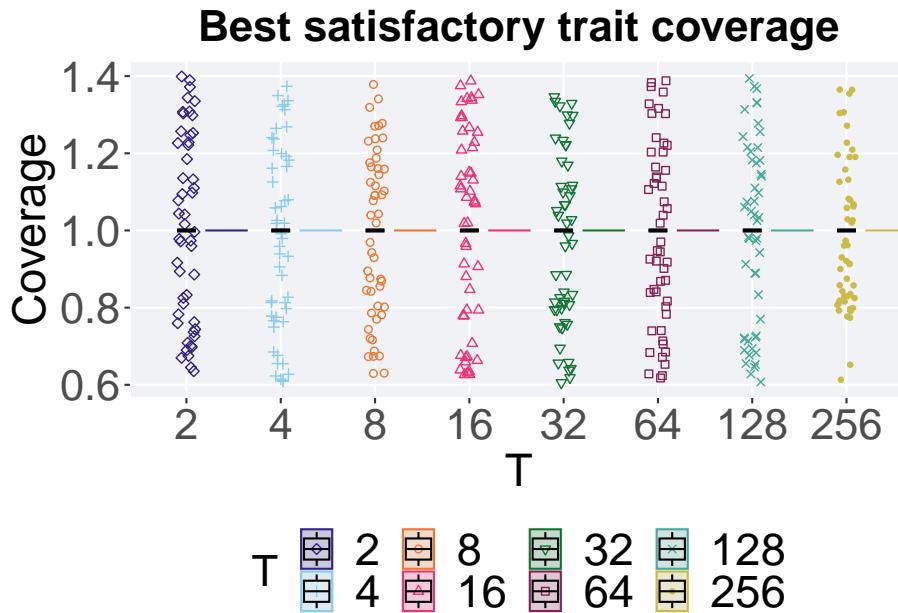
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage"
  ) +
  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
```



7.3.1.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
filter(tor_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = T, y = val, color = T, fill = T, shape = T)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.0) +
  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"
  ) +
  scale_x_discrete(
    name="T"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best satisfactory trait coverage") +
  p_theme
```



7.3.1.2.1 Stats

Summary statistics for the best satisfactory trait coverage throughout 50,000 generations.

```
coverage = filter(tor_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives')
group_by(coverage, T) %>%
  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)
  )

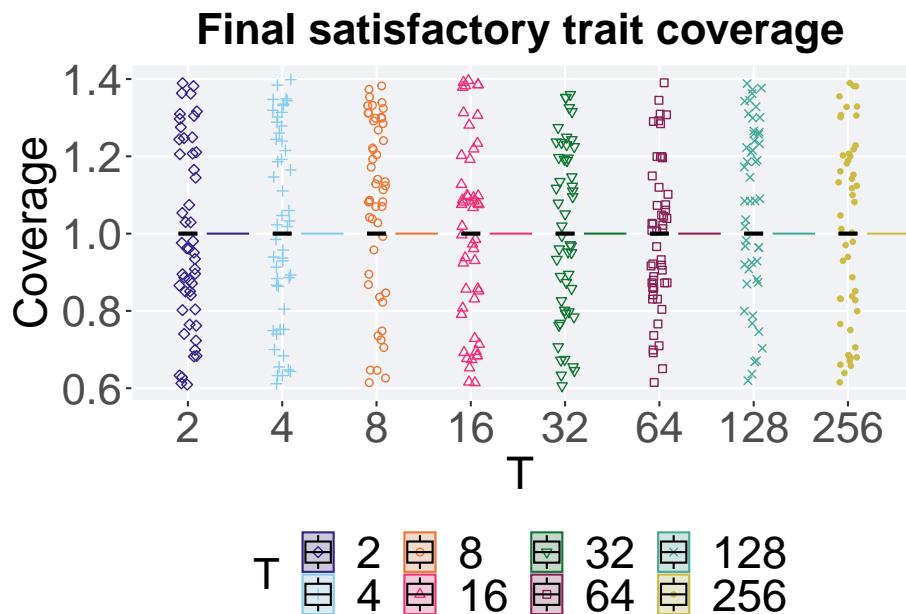
## # A tibble: 8 x 8
##   T      count na_cnt  min median  mean  max  IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2       50     0     1     1     1     1     0
## 2 4       50     0     1     1     1     1     0
## 3 8       50     0     1     1     1     1     0
## 4 16      50     0     1     1     1     1     0
## 5 32      50     0     1     1     1     1     0
## 6 64      50     0     1     1     1     1     0
```

```
## 7 128      50     0     1     1     1     1     0
## 8 256      50     0     1     1     1     1     0
```

7.3.1.3 End of 50,000 generations

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

```
filter(tor_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = T, y = pop_uni_obj, color = T, fill = T, shape = T)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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"
  ) +
  scale_x_discrete(
    name = "T"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final satisfactory trait coverage") +
  p_theme
```



7.3.1.3.1 Stats

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

```
coverage = filter(tor_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, T) %>%
  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
##   T      count na_cnt  min median  mean  max  IQR
##   <fct> <int>   <int> <dbl> <dbl> <int> <dbl>
## 1 2        50      0     1     1     1     1     0
## 2 4        50      0     1     1     1     1     0
## 3 8        50      0     1     1     1     1     0
## 4 16       50      0     1     1     1     1     0
## 5 32       50      0     1     1     1     1     0
## 6 64       50      0     1     1     1     1     0
## 7 128      50      0     1     1     1     1     0
## 8 256      50      0     1     1     1     1     0
```

7.3.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the contradictory objectives diagnostic.

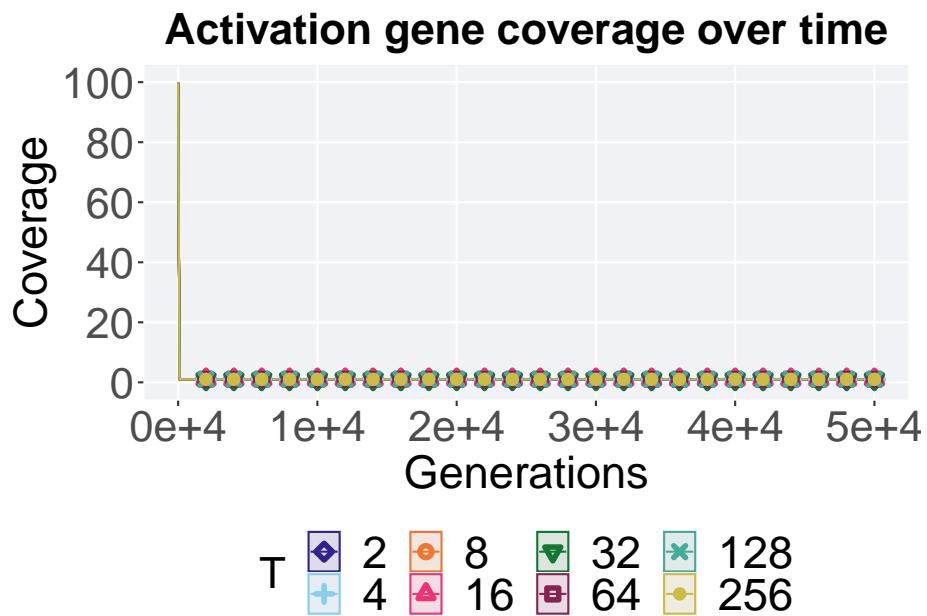
7.3.2.1 Coverage over time

Activation gene coverage over time.

```
lines = filter(tor_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

## `summarise()` has grouped output by 'T'. You can override using the `.groups` argument.
```

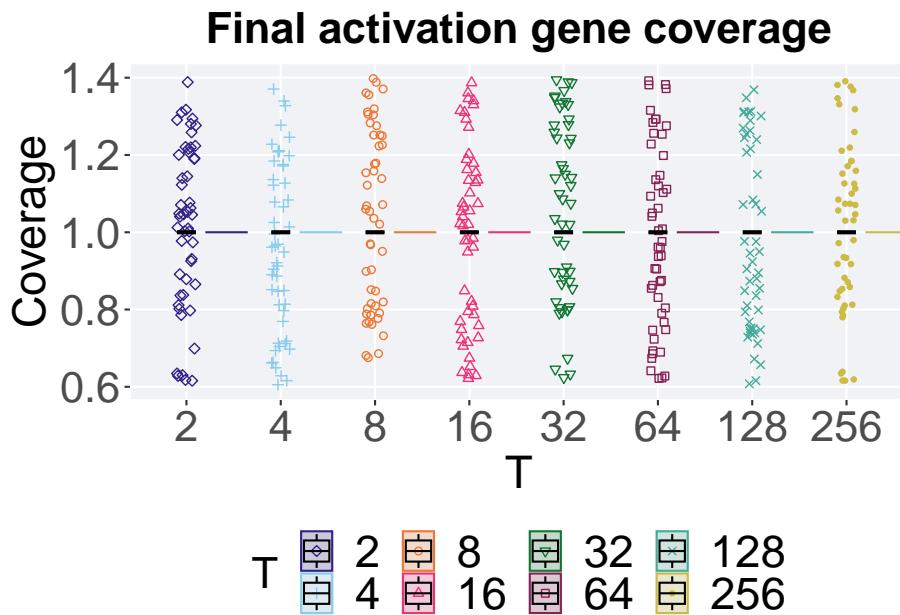
```
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2)
  scale_y_continuous(
    name="Coverage",
    limits=c(-1, 101),
    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
```



7.3.2.2 End of 50,000 generations

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

```
filter(tor_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = T, y = uni_str_pos, color = T, fill = T, shape = T)) +
  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"
  ) +
  scale_x_discrete(
    name = "T"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final activation gene coverage") +
  p_theme
```



7.3.2.2.1 Stats

Summary statistics for activation gene coverage in the population at the end of 50,000 generations.

```

coverage = filter(tor_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, T) %>%
  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
##   T      count na_cnt  min median  mean  max  IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 2        50     0     1     1     1     1     0
## 2 4        50     0     1     1     1     1     0
## 3 8        50     0     1     1     1     1     0
## 4 16       50     0     1     1     1     1     0
## 5 32       50     0     1     1     1     1     0
## 6 64       50     0     1     1     1     1     0
## 7 128      50     0     1     1     1     1     0
## 8 256      50     0     1     1     1     1     0

```

7.3.3 Multi-valley crossing

7.3.3.1 Satisfactory trait coverage over time

```

lines = filter(tor_ot_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

## `summarise()` has grouped output by 'T'. You can override using the `.groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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"
  ) +

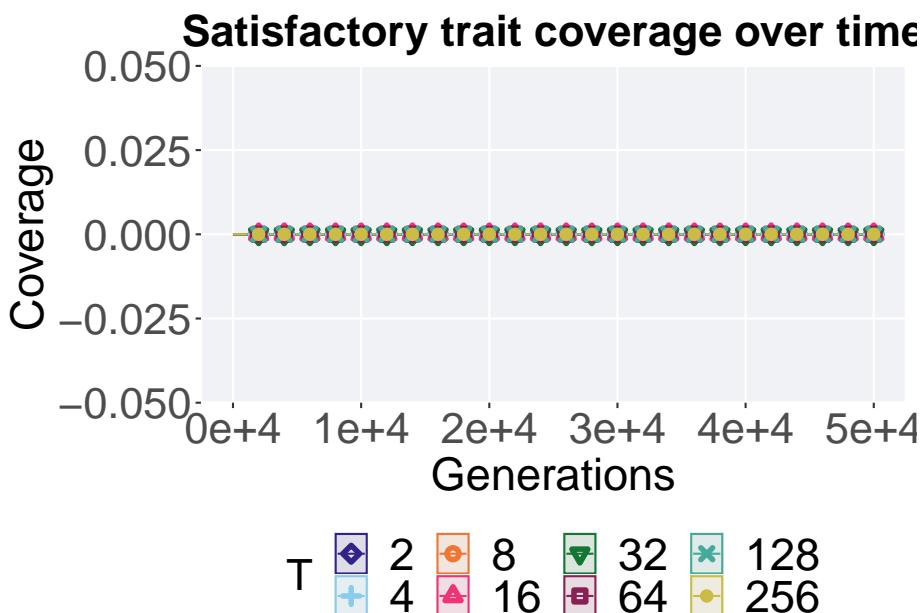
```

```

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

```



7.3.3.2 Satisfactory trait coverage comparison

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

```

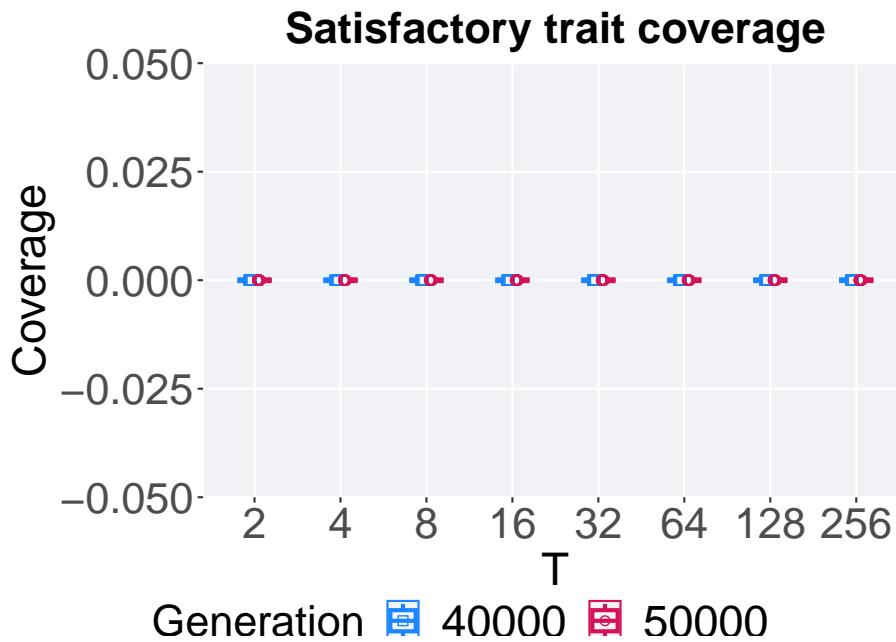
# 80% and final generation comparison
end = filter(tor_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 50000)
end$Generation <- factor(end$gen)

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

mvc_p = ggplot(mid, aes(x = T, y=pop_uni_obj, group = T, shape = Generation)) +

```

```
geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -.15), lwd = 0.7, col = mvc_col[1]) +  
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1]) +  
  geom_point(data = end, aes(x = T, y=pop_uni_obj), col = mvc_col[2], position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2]) +  
  geom_boxplot(data = end, aes(x = T, y=pop_uni_obj), position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2]) +  
  
  scale_y_continuous(  
    name="Coverage",  
  ) +  
  scale_x_discrete(  
    name="T"  
  ) +  
  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") +  
    theme(legend.position="none"),  
    legend,  
    nrow=2,  
    rel_heights = c(1,.05),  
    label_size = TSIZE  
)
```



7.3.3.2.1 Stats

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

```
slices = filter(tor_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, 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 'T'. You can override using the `.groups` argument.

## # A tibble: 16 x 9
## # Groups:   T [8]
##   T     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <int> <dbl> <dbl> <int> <dbl>
```

```

##  1 2      50000      50      0      0      0      0      0      0
##  2 2      40000      50      0      0      0      0      0      0
##  3 4      50000      50      0      0      0      0      0      0
##  4 4      40000      50      0      0      0      0      0      0
##  5 8      50000      50      0      0      0      0      0      0
##  6 8      40000      50      0      0      0      0      0      0
##  7 16     50000      50      0      0      0      0      0      0
##  8 16     40000      50      0      0      0      0      0      0
##  9 32     50000      50      0      0      0      0      0      0
## 10 32    40000      50      0      0      0      0      0      0
## 11 64     50000      50      0      0      0      0      0      0
## 12 64     40000      50      0      0      0      0      0      0
## 13 128    50000      50      0      0      0      0      0      0
## 14 128    40000      50      0      0      0      0      0      0
## 15 256    50000      50      0      0      0      0      0      0
## 16 256    40000      50      0      0      0      0      0      0

T 2

wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 2 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_uni_obj and filter(slices,
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 4

wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 4 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_uni_obj and filter(slices,
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 8

wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 8 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##

```

```
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_uni_obj and filter(slices, T == 8 & Ge
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 16 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_uni_obj and filter(slices, T == 16 &
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 32 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_uni_obj and filter(slices, T == 32 &
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 64
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 64 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_uni_obj and filter(slices, T == 64 &
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

T 128
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, T == 128 & Generation == 40000)$pop_uni_obj,
             alternative = 't')
```

```

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 128 & Generation == 50000)$pop_uni_obj and filter(slices  

## W = 1250, p-value = NA  

## alternative hypothesis: true location shift is not equal to 0  

T 256  

wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_uni_obj,  

            y = filter(slices, T == 256 & Generation == 40000)$pop_uni_obj,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 256 & Generation == 50000)$pop_uni_obj and filter(slices  

## W = 1250, p-value = NA  

## alternative hypothesis: true location shift is not equal to 0

```

7.3.3.3 Activation gene coverage over time

```

lines = filter(tor_ot_mvc, diagnostic == 'contradictory_objectives') %>%  

  group_by(T, gen) %>%  

  dplyr::summarise(  

    min = min(uni_str_pos),  

    mean = mean(uni_str_pos),  

    max = max(uni_str_pos)
  )  

## `summarise()` has grouped output by 'T'. You can override using the `groups`  

## argument.  

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +  

  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"
  ) +  

  scale_x_continuous(  

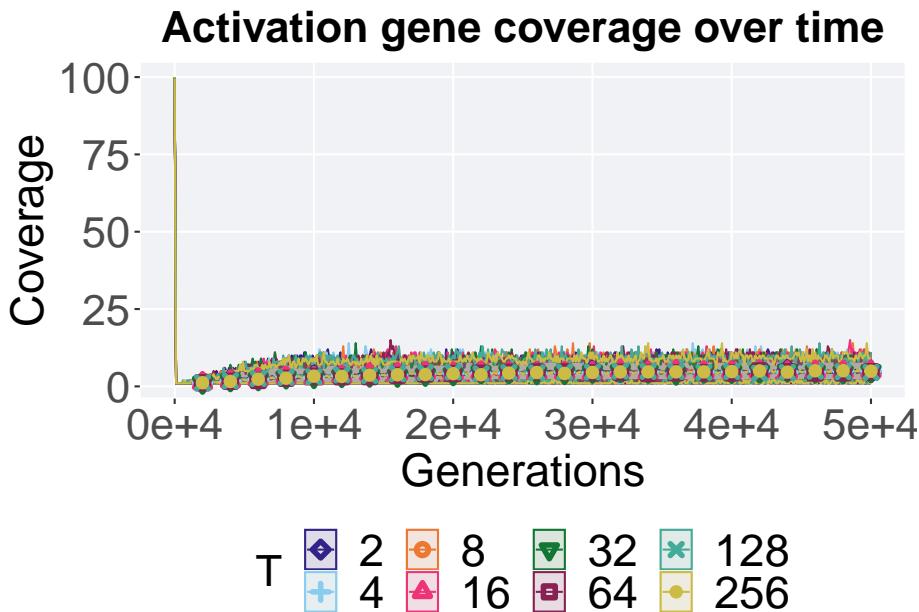
    name="Generations",
    limits=c(0, 50000),
    breaks=c(0, 10000, 20000, 30000, 40000, 50000),
    labels=c("0e+4", "1e+4", "2e+4", "3e+4", "4e+4", "5e+4")
  ) +

```

```

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

```



7.3.3.4 Activation gene coverage comparison

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

```

# 80% and final generation comparison
end = filter(tor_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 50000)
end$Generation <- factor(end$gen)

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

mvc_p = ggplot(mid, aes(x = T, y=uni_str_pos, group = T, shape = Generation)) +
  geom_point(col = mvc_col[1], position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), nudge.y = 0),
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = T, y=uni_str_pos), col = mvc_col[2], position = position_jitternudge(jitter.width = .03, nudge.x = 0.15), nudge.y = 0),
  geom_boxplot(data = end, aes(x = T, y=uni_str_pos), position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

  scale_y_continuous(

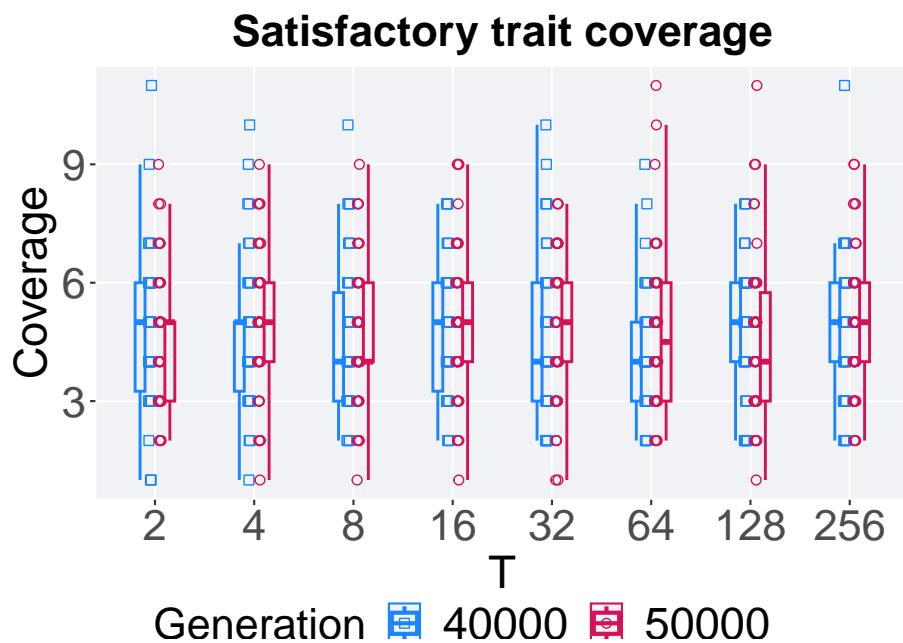
```

```

    name="Coverage",
) +
scale_x_discrete(
  name="T"
) +
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") +
  theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



7.3.3.4.1 Stats

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

```

slices = filter(tor_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, 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 'T'. You can override using the ` `.groups` argument.

## # A tibble: 16 x 9
## # Groups: T [8]
##   T     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 2     50000      50     0     2     5     4.6     9     2
## 2 2     40000      50     0     1     5     4.7    11    2.75
## 3 4     50000      50     0     1     5     4.86    9     2
## 4 4     40000      50     0     1     5     4.68    10    1.75
## 5 8     50000      50     0     1     4     4.64    9     2
## 6 8     40000      50     0     2     4     4.54    10    2.75
## 7 16    50000      50     0     1     5     4.92    9     2
## 8 16    40000      50     0     2     5     4.74    8     2.75
## 9 32    50000      50     0     1     5     4.86    8     2
## 10 32   40000      50     0     2     4     4.6     10    3
## 11 64    50000      50     0     2     4.5    4.74    11    3
## 12 64    40000      50     0     2     4     4.34    9     2
## 13 128   50000      50     0     1     4     4.52    11    2.75
## 14 128   40000      50     0     2     5     4.78    8     2
## 15 256   50000      50     0     2     5     4.92    9     2
## 16 256   40000      50     0     2     5     4.78    11    2

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 2 & Generation == 40000)$uni_str_pos,
             alternative = 't')

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

```

```

## data: filter(slices, T == 2 & Generation == 50000)$uni_str_pos and filter(slices, T == 4 & Generation == 50000)$uni_str_pos
## W = 1205, p-value = 0.7547
## alternative hypothesis: true location shift is not equal to 0

T 4
wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 4 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$uni_str_pos and filter(slices, T == 8 & Generation == 50000)$uni_str_pos
## W = 1355, p-value = 0.4625
## alternative hypothesis: true location shift is not equal to 0

T 8
wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 8 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$uni_str_pos and filter(slices, T == 16 & Generation == 50000)$uni_str_pos
## W = 1351.5, p-value = 0.4781
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 16 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$uni_str_pos and filter(slices, T == 32 & Generation == 50000)$uni_str_pos
## W = 1310, p-value = 0.677
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 32 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```

##  

## data: filter(slices, T == 32 & Generation == 50000)$uni_str_pos and filter(slices, T == 32 &  

## W = 1383, p-value = 0.3552  

## alternative hypothesis: true location shift is not equal to 0  

T 64  

wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, T == 64 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 64 & Generation == 50000)$uni_str_pos and filter(slices, T == 64 &  

## W = 1373.5, p-value = 0.3871  

## alternative hypothesis: true location shift is not equal to 0  

T 128  

wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, T == 128 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 128 & Generation == 50000)$uni_str_pos and filter(slices, T == 128 &  

## W = 1078, p-value = 0.2303  

## alternative hypothesis: true location shift is not equal to 0  

T 256  

wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, T == 256 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, T == 256 & Generation == 50000)$uni_str_pos and filter(slices, T == 256 &  

## W = 1280, p-value = 0.8358  

## alternative hypothesis: true location shift is not equal to 0

```

7.4 Multi-path exploration results

Here we present the results for **best performances** and **activation gene coverage** found by each tournament selection size value replicate on the multi-path exploration diagnostic. Best performance found refers to the largest average

trait score found in a given population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both values fall between 0 and 100.

7.4.1 Performance

Here we analyze the performances for each parameter replicate on the multi-path exploration diagnostic.

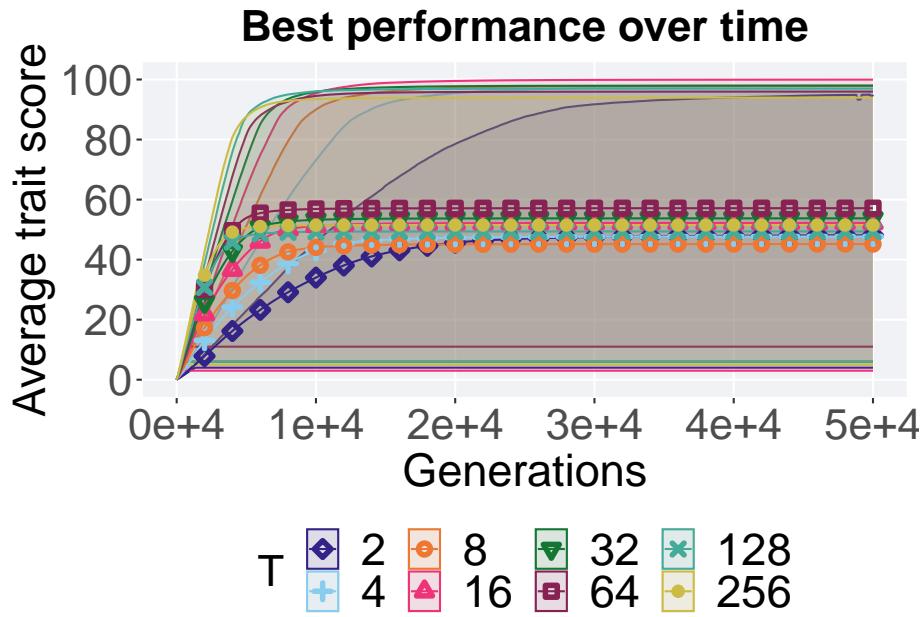
7.4.1.1 Performance over time

Performance over time.

```
lines = filter(tor_ot, diagnostic == 'multipath_exploration') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

## `summarise()` has grouped output by 'T'. You can override using the `.groups` argument.

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = T, fill = T, shape = T,
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2),
  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(-1, 101),
    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("Best performance over time") +
  p_theme
```



7.4.1.2 Best performance throughout

Here we plot the performance of the best performing solution found throughout 50,000 generations.

```
performance = filter(tor_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = T, y = val / DIMENSIONALITY, color = T, fill = T, shape = T)) +
  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="T"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  p_theme
```

7.4.1.2.1 Stats

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

```
performance = filter(tor_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration')
group_by(performance, T) %>%
  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
##   T      count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2        50     0     4    52.5   48.1   95.0   43.4
## 2 4        50     0     5    45.0   47.7   97.8   58.2
## 3 8        50     0    6.00   42.5   45.2   97.9   53.2
## 4 16       50     0     3    48.0   52.2  100.    61.5
## 5 32       50     0     6    53.0   53.8   98.0   55.2
## 6 64       50     0    11    58.0   57.1   96.0   44.0
## 7 128      50     0    6.00   52.0   49.4   97.0   47.0
## 8 256      50     0     5    51.0   51.5   94.0   41.5
```

Kruskal–Wallis test provides evidence of no statistical differences among the best performing solution found throughout 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: val by T
## Kruskal-Wallis chi-squared = 6.9356, df = 7, p-value = 0.4356
```

7.4.1.3 End of 50,000 generations

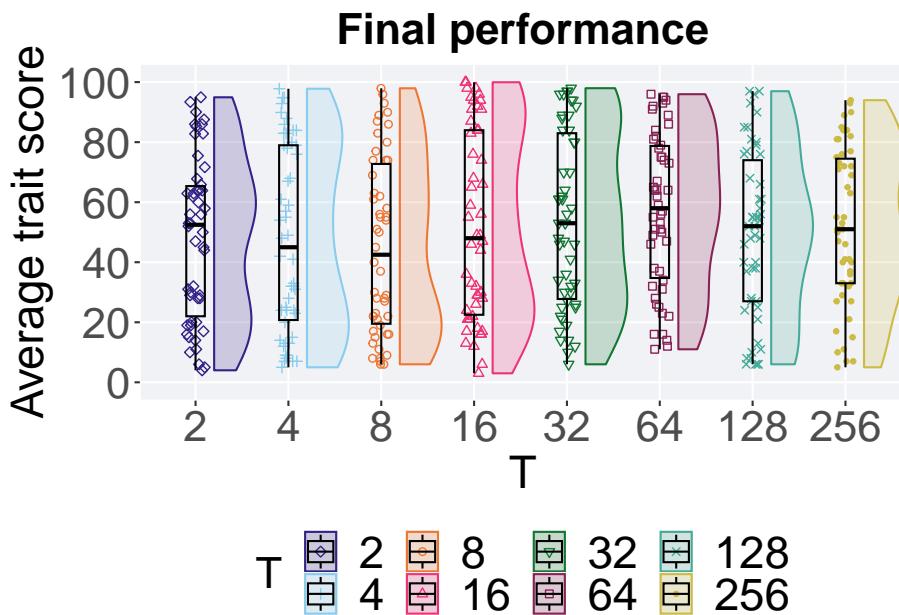
Best performance in the population at the end of 50,000 generations.

```
filter(tor_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = T, y = pop_fit_max / DIMENSIONALITY, color = T, fill = T, shape = T))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5)
  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="T"
) +
scale_shape_manual(values=SHAPE) +
scale_colour_manual(values = cb_palette) +
scale_fill_manual(values = cb_palette) +
ggtitle("Final performance") +
p_theme

```



7.4.1.3.1 Stats

Summary statistics for the best performance in the population at the end of 50,000 generations.

```

performance = filter(tor_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(performance, T) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max / DIMENSIONALITY)),
    min = min(pop_fit_max / DIMENSIONALITY, na.rm = TRUE),
    max = max(pop_fit_max / DIMENSIONALITY),
    mean = mean(pop_fit_max / DIMENSIONALITY),
    median = median(pop_fit_max / DIMENSIONALITY)
  )

```

```

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
##   T      count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2        50     0     4    52.5  48.1  94.9  43.4
## 2 4        50     0     5    45.0  47.7  97.8  58.2
## 3 8        50     0    6.00   42.5  45.2  97.9  53.2
## 4 16       50     0     3    48.0  52.2 100.   61.5
## 5 32       50     0     6    53.0  53.8  98.0  55.2
## 6 64       50     0    11    58.0  57.1  96.0  44.0
## 7 128      50     0    6.00   52.0  49.4  97.0  47.0
## 8 256      50     0     5    51.0  51.5  94.0  41.5

```

Kruskal–Wallis test provides evidence of statistical differences among best performance in the population at the end of 50,000 generations.

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

```

##
## Kruskal-Wallis rank sum test
##
## data: pop_fit_max by T
## Kruskal-Wallis chi-squared = 6.9356, df = 7, p-value = 0.4356

```

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

```

pairwise.wilcox.test(x = performance$pop_fit_max, g = performance$T , p.adjust.method =
                      paired = FALSE, conf.int = FALSE, alternative = 'l')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$pop_fit_max and performance$T
##
##      2 4 8 16 32 64 128
## 4  1 - - - - - -
## 8  1 1 - - - - -
## 16 1 1 1 - - - -
## 32 1 1 1 1 - - -
## 64 1 1 1 1 1 - -
## 128 1 1 1 1 1 1 -
## 256 1 1 1 1 1 1 1

```

```
##  
## P value adjustment method: bonferroni
```

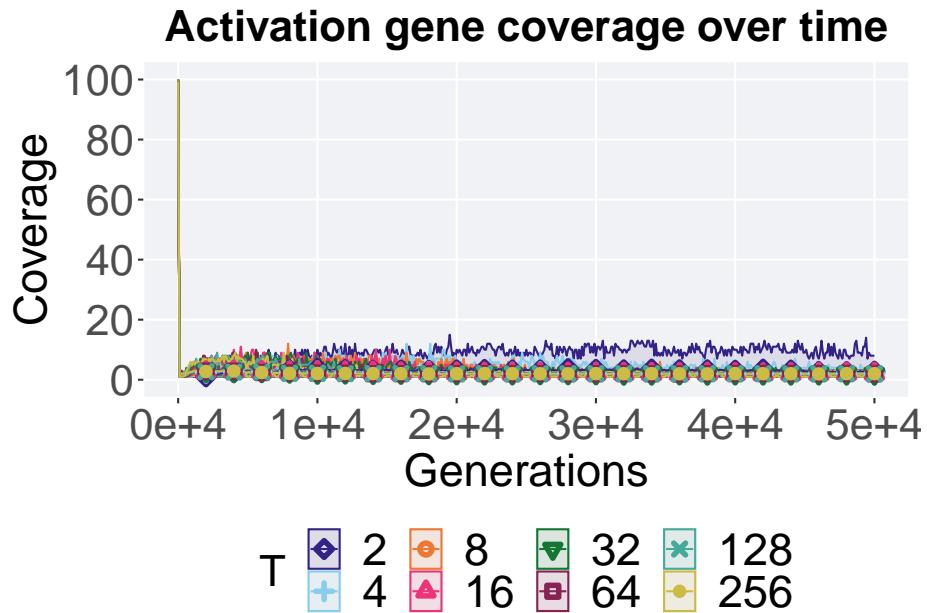
7.4.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the multi-path exploration diagnostic.

7.4.2.1 Coverage over time

Activation gene coverage over time.

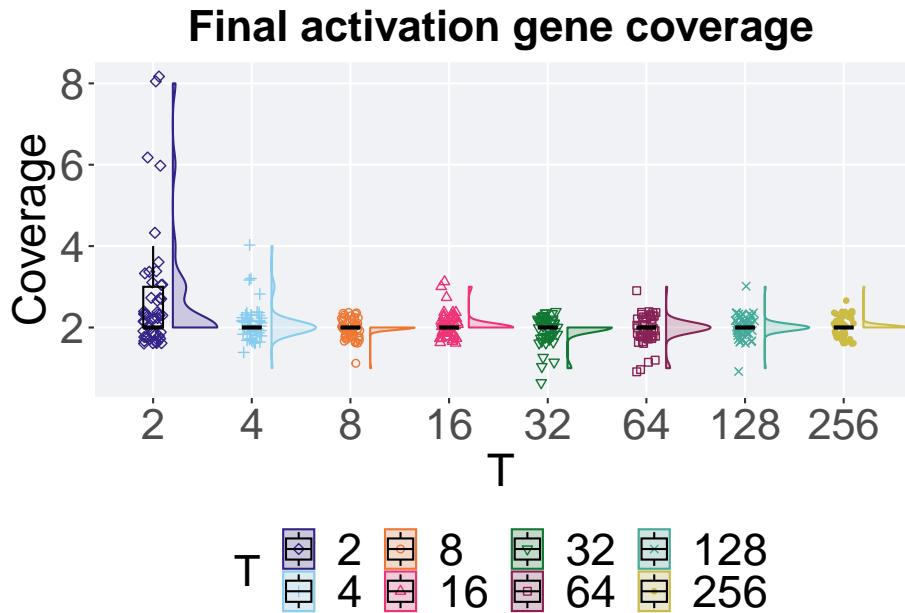
```
lines = filter(tor_ot, diagnostic == 'multipath_exploration') %>%  
  group_by(T, gen) %>%  
  dplyr::summarise(  
    min = min(uni_str_pos),  
    mean = mean(uni_str_pos),  
    max = max(uni_str_pos)  
  )  
  
## `summarise()` has grouped output by 'T'. You can override using the `.groups`  
## argument.  
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +  
  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(-1, 101),  
    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
```



7.4.2.2 End of 50,000 generations

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

```
filter(tor_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = T, y = uni_str_pos, color = T, fill = T, shape = T)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.0) +
  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"
  ) +
  scale_x_discrete(
    name="T"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final activation gene coverage") +
  p_theme
```



7.4.2.2.1 Stats

Summary statistics for the activation gene coverage in the population at the end of 50,000 generations.

```
coverage = filter(tor_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(coverage, T) %>%
  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
##   T      count na_cnt  min median  mean  max  IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 2        50     0     2     2  2.68     8     1
## 2 4        50     0     1     2  2.08     4     0
## 3 8        50     0     1     2  1.98     2     0
## 4 16       50     0     2     2  2.06     3     0
## 5 32       50     0     1     2  1.92     2     0
## 6 64       50     0     1     2  1.94     3     0
```

```
## 7 128      50      0      1      2      2      3      0
## 8 256      50      0      2      2     2.02      3      0
```

Kruskal–Wallis test provides evidence of statistical differences among activation gene coverage in the population at the end of 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: uni_str_pos by T
## Kruskal-Wallis chi-squared = 61.183, df = 7, p-value = 8.757e-11
```

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

```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$T , p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')

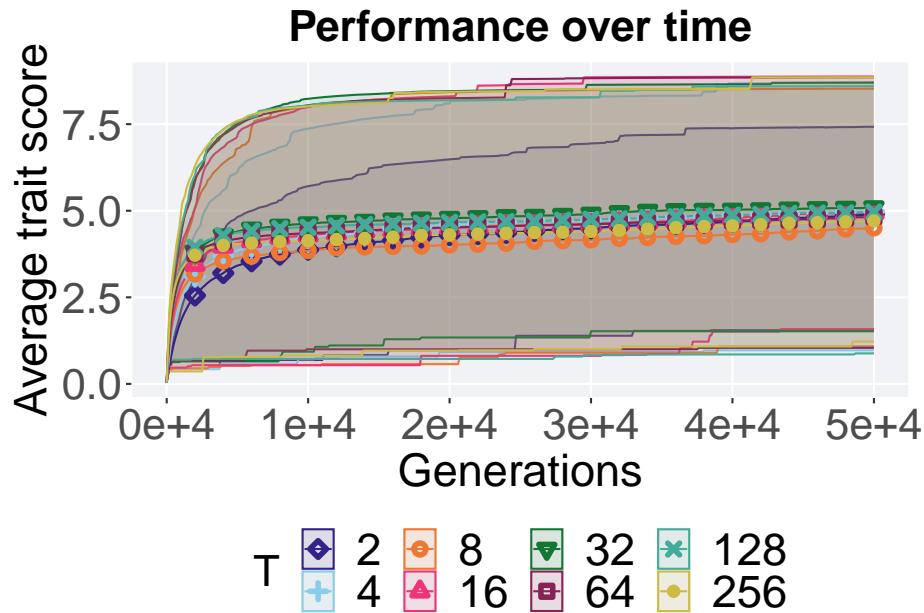
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$uni_str_pos and coverage$T
##
##    2      4      8      16      32      64      128
## 4  0.04907 -      -      -      -      -      -
## 8  0.00031 1.00000 -      -      -      -      -
## 16 0.02131 1.00000 1.00000 -      -      -      -
## 32 0.00011 0.58051 1.00000 0.24142 -      -      -
## 64 0.00044 1.00000 1.00000 0.99216 1.00000 -      -
## 128 0.00134 1.00000 1.00000 1.00000 1.00000 1.00000 -
## 256 0.00191 1.00000 1.00000 1.00000 0.70887 1.00000 1.00000
##
## P value adjustment method: bonferroni
```

7.4.3 Multi-valley crossing

7.4.3.1 Performance over time

```
# data for lines and shading on plots
lines = filter(tor_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(T, 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 'T'. You can override using the `groups`  
## argument.  
ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +  
  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 =  
    1) +  
  scale_y_continuous(  
    name="Average trait score"  
) +  
  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 +  
  guides(  
    shape=guide_legend(nrow=2, title.position = "left"),  
    color=guide_legend(nrow=2, title.position = "left"),  
    fill=guide_legend(nrow=2, title.position = "left")  
)
```



7.4.3.2 Performance comparison

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

```
# 80% and final generation comparison
end = filter(tor_ot_mvc, diagnostic == 'multipath_exploration' & gen == 50000 & T != '1')
end$Generation <- factor(end$gen)

mid = filter(tor_ot_mvc, diagnostic == 'multipath_exploration' & gen == 40000 & T != '1')
mid$Generation <- factor(mid$gen)

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

  geom_point(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], size = 1)
  geom_boxplot(data = end, aes(x = T, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 1), lwd = 0.7, col = mvc_col[2])

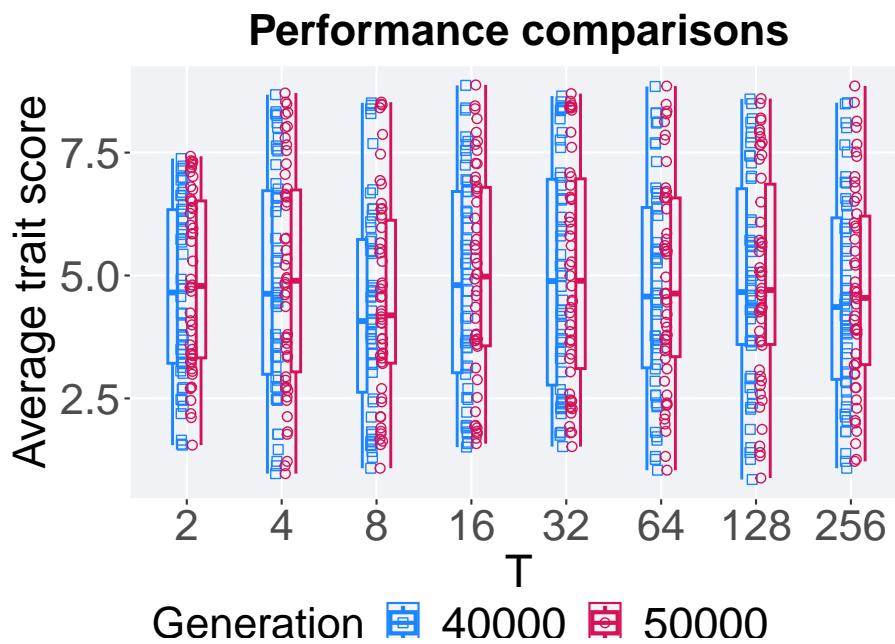
  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="T"
  ) +
  scale_shape_manual(values=c(0,1)) +
```

```

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
)

```



7.4.3.3 Stats

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

```

slices = filter(tor_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_fit_max / DIMENSIONALITY)),
    mean = mean(pop_fit_max / DIMENSIONALITY),
    min = min(pop_fit_max / DIMENSIONALITY),
    max = max(pop_fit_max / DIMENSIONALITY),
    median = median(pop_fit_max / DIMENSIONALITY),
    stdev = sd(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 'T'. You can override using the `groups` argument.

## # A tibble: 16 x 9
## # Groups:   T [8]
##   T     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>      <int> <int> <dbl> <dbl> <dbl> <dbl>
## 1 2     50000       50    0 1.55   4.79  4.89  7.43  3.19
## 2 2     40000       50    0 1.55   4.65  4.68  7.38  3.13
## 3 4     50000       50    0 0.970  4.89  4.98  8.71  3.70
## 4 4     40000       50    0 0.970  4.63  4.87  8.68  3.74
## 5 8     50000       50    0 1.08   4.19  4.50  8.53  2.91
## 6 8     40000       50    0 1.08   4.07  4.31  8.51  3.11
## 7 16    50000       50    0 1.58   4.98  4.94  8.88  3.22
## 8 16    40000       50    0 1.51   4.80  4.80  8.87  3.69
## 9 32    50000       50    0 1.52   4.89  5.09  8.70  3.86
## 10 32   40000       50    0 1.52   4.89  5.02  8.65  4.19
## 11 64    50000       50    0 1.04   4.63  4.84  8.85  3.23
## 12 64    40000       50    0 1.04   4.57  4.71  8.85  3.26
## 13 128   50000       50    0 0.880  4.70  4.96  8.60  3.26
## 14 128   40000       50    0 0.850  4.66  4.88  8.59  3.17
## 15 256   50000       50    0 1.22   4.54  4.70  8.86  3.02
## 16 256   40000       50    0 1.08   4.36  4.57  8.51  3.29

T 2
wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 2 & Generation == 50000)$pop_fit_max and filter(slices, T == 2 & Generation == 40000)$pop_fit_max
## W = 1354, p-value = 0.4755
## alternative hypothesis: true location shift is not equal to 0

T 4
wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 4 & Generation == 40000)$pop_fit_max,
```

```
alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 4 & Generation == 50000)$pop_fit_max and filter(slices, T == 4 & Ge
## W = 1305, p-value = 0.7071
## alternative hypothesis: true location shift is not equal to 0

T 8
wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 8 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$pop_fit_max and filter(slices, T == 8 & Ge
## W = 1339, p-value = 0.5418
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 16 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$pop_fit_max and filter(slices, T == 16 &
## W = 1322, p-value = 0.6221
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$pop_fit_max,
             y = filter(slices, T == 32 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$pop_fit_max and filter(slices, T == 32 &
## W = 1298, p-value = 0.7433
## alternative hypothesis: true location shift is not equal to 0

T 64
```

```
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 64 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1314, p-value = 0.6616
## alternative hypothesis: true location shift is not equal to 0

T 128

wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 128 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 128 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1304, p-value = 0.7123
## alternative hypothesis: true location shift is not equal to 0

T 256

wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$pop_fit_max,
            y = filter(slices, T == 256 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 256 & Generation == 50000)$pop_fit_max and filter(slices
## W = 1316.5, p-value = 0.6491
## alternative hypothesis: true location shift is not equal to 0
```

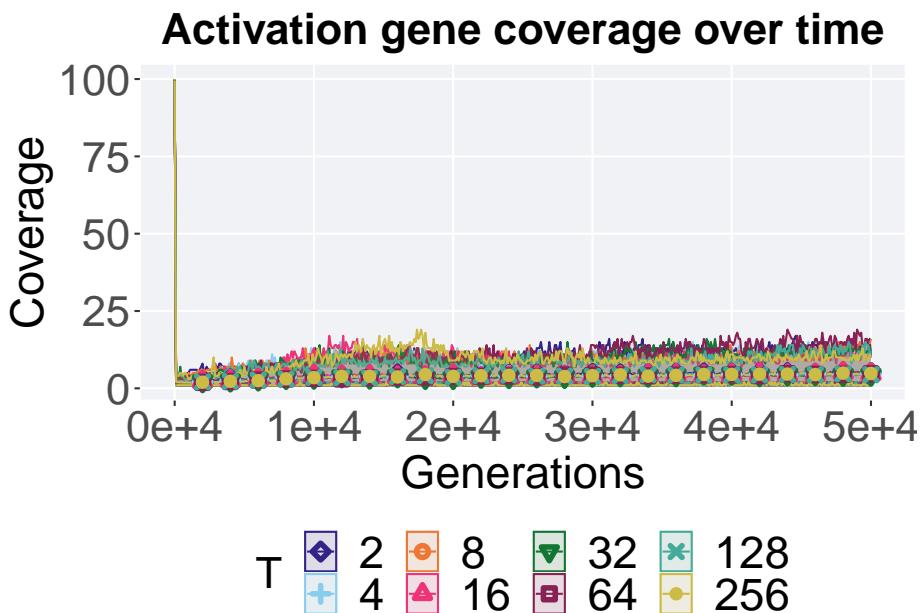
7.4.3.4 Activation gene coverage over time

```
lines = filter(tor_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(T, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

## `summarise()` has grouped output by 'T'. You can override using the `groups`
```

```
## argument.

ggplot(lines, aes(x=gen, y=mean, group = T, fill = T, color = T, shape = T)) +
  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"
  ) +
  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
```



7.4.3.5 Activation gene coverage comparison

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

```

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

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

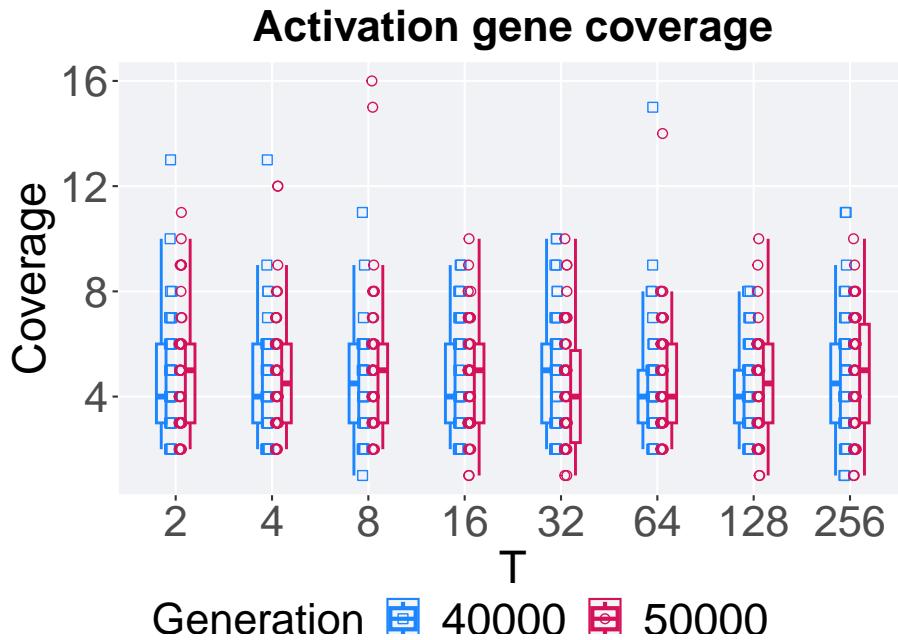
mvc_p = ggplot(mid, aes(x = T, y=uni_str_pos, group = T, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nuc
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1]

  geom_point(data = end, aes(x = T, y=uni_str_pos), col = mvc_col[2], position = positio
  geom_boxplot(data = end, aes(x = T, y=uni_str_pos), position = position_nudge(x = .15

  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="T"
  )+
  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") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



7.4.3.5.1 Stats

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

```
slices = filter(tor_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(T, 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 'T'. You can override using the `.`groups` argument.

## # A tibble: 16 x 9
## # Groups:   T [8]
##   T     Generation count  na_cnt   min  median   mean   max    IQR
##   <fct> <fct>     <int>  <int>  <int>  <dbl>  <dbl> <int>  <dbl>
```

```

##   1 2      50000      50      0      2      5    4.94    11    3
##   2 2      40000      50      0      2      4    4.76    13    3
##   3 4      50000      50      0      2      4.5   4.8     12    3
##   4 4      40000      50      0      2      4    4.48    13    3
##   5 8      50000      50      0      2      5    5.1     16    3
##   6 8      40000      50      0      1      4.5   4.38    11    3
##   7 16     50000      50      0      1      5    4.68    10    3
##   8 16     40000      50      0      2      4    4.4     9     3
##   9 32     50000      50      0      1      4    4.34    10   3.5
##  10 32    40000      50      0      2      5    4.66    10    3
##  11 64     50000      50      0      2      4    4.72    14    3
##  12 64     40000      50      0      2      4    4.44    15    2
##  13 128    50000      50      0      1      4.5   4.4     10    3
##  14 128    40000      50      0      2      4    4.38     8    2
##  15 256    50000      50      0      1      5    4.82    10   3.75
##  16 256    40000      50      0      1      4.5   4.5     11    3

T 2

wilcox.test(x = filter(slices, T == 2 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 2 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
##  Wilcoxon rank sum test with continuity correction
##
##  data:  filter(slices, T == 2 & Generation == 50000)$uni_str_pos and filter(slices,
##  W = 1272.5, p-value = 0.878
##  alternative hypothesis: true location shift is not equal to 0

T 4

wilcox.test(x = filter(slices, T == 4 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 4 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
##  Wilcoxon rank sum test with continuity correction
##
##  data:  filter(slices, T == 4 & Generation == 50000)$uni_str_pos and filter(slices,
##  W = 1347.5, p-value = 0.498
##  alternative hypothesis: true location shift is not equal to 0

T 8

wilcox.test(x = filter(slices, T == 8 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 8 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##

```

```
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 8 & Generation == 50000)$uni_str_pos and filter(slices, T == 8 & Ge
## W = 1396, p-value = 0.3094
## alternative hypothesis: true location shift is not equal to 0

T 16
wilcox.test(x = filter(slices, T == 16 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 16 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 16 & Generation == 50000)$uni_str_pos and filter(slices, T == 16 &
## W = 1386.5, p-value = 0.3422
## alternative hypothesis: true location shift is not equal to 0

T 32
wilcox.test(x = filter(slices, T == 32 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 32 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 32 & Generation == 50000)$uni_str_pos and filter(slices, T == 32 &
## W = 1155.5, p-value = 0.5116
## alternative hypothesis: true location shift is not equal to 0

T 64
wilcox.test(x = filter(slices, T == 64 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 64 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, T == 64 & Generation == 50000)$uni_str_pos and filter(slices, T == 64 &
## W = 1346.5, p-value = 0.5018
## alternative hypothesis: true location shift is not equal to 0

T 128
wilcox.test(x = filter(slices, T == 128 & Generation == 50000)$uni_str_pos,
             y = filter(slices, T == 128 & Generation == 40000)$uni_str_pos,
             alternative = 't')
```

```
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, T == 128 & Generation == 50000)$uni_str_pos and filter(slices  
## W = 1260.5, p-value = 0.9443  
## alternative hypothesis: true location shift is not equal to 0  
T 256  
wilcox.test(x = filter(slices, T == 256 & Generation == 50000)$uni_str_pos,  
            y = filter(slices, T == 256 & Generation == 40000)$uni_str_pos,  
            alternative = 't')  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, T == 256 & Generation == 50000)$uni_str_pos and filter(slices  
## W = 1377.5, p-value = 0.3761  
## alternative hypothesis: true location shift is not equal to 0
```

Chapter 8

Genotypic fitness sharing

We present the results from our parameter sweep on genotypic fitness sharing. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, no similarity penalty is used, and only stochastic remainder selection is used to identify parent solutions.

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

8.1 Exploitation rate results

Here we present the results for **best performances** found by each genotypic fitness sharing sigma value replicate on the exploitation rate diagnostic. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, no similarity penalty is used, and only stochastic remainder selection is used to identify parent solutions.

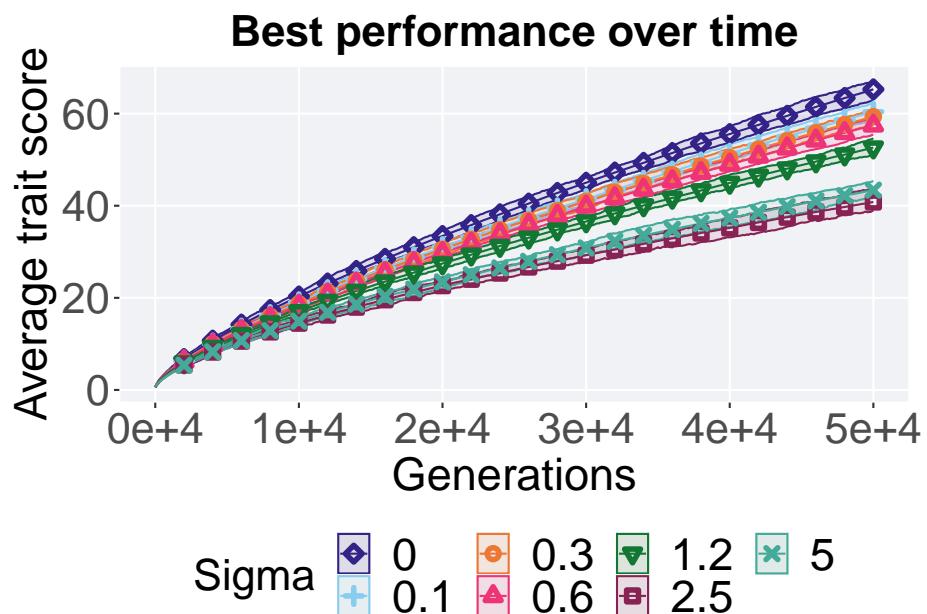
8.1.1 Performance over time

Performance over time.

```
lines = filter(gfs_ot, diagnostic == 'exploitation_rate') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )
```

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

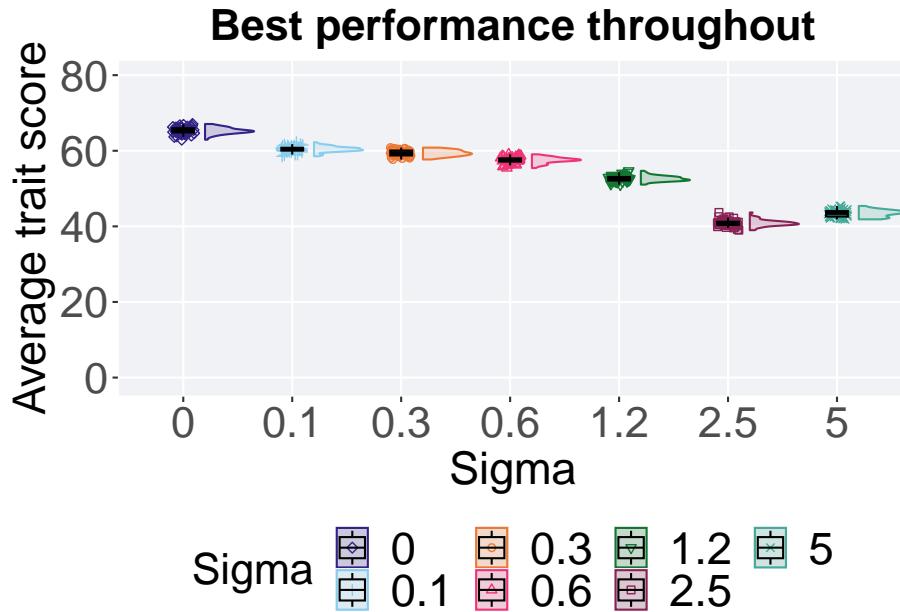
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color =
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2),
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2)
  scale_y_continuous(
    name="Average trait score"
  ) +
  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("Best performance over time") +
  p_theme
```



8.1.2 Best performance throughout

The best performance found throughout 50,000 generations.

```
filter(gfs_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
    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, 80),
      breaks=seq(0,80, 20),
      labels=c("0", "20", "40", "60", "80")
    ) +
    scale_x_discrete(
      name="Sigma"
    ) +
    scale_shape_manual(values=SHAPE) +
    scale_colour_manual(values = cb_palette) +
    scale_fill_manual(values = cb_palette) +
    ggtitle("Best performance throughout") +
    p_theme
```



8.1.2.1 Stats

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

```
performance = filter(gfs_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate'
group_by(performance, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0         50      0  62.9   65.3   65.3  67.1  1.11
## 2 0.1       50      0  58.5   60.4   60.4  62.2  0.717
## 3 0.3       50      0  57.7   59.3   59.4  60.8  1.31
## 4 0.6       50      0  55.4   57.6   57.5  59.1  0.843
## 5 1.2       50      0  51.1   52.5   52.6  54.7  0.907
## 6 2.5       50      0  39.0   40.8   40.9  43.7  0.813
## 7 5         50      0  41.9   43.6   43.4  45.4  1.43
```

Kruskal–Wallis test provides evidence of significant differences among sigma values on the best performance found throughout 50,000 generations

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

##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 336.49, df = 6, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performance found throughout 50,000 generations.

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

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
```

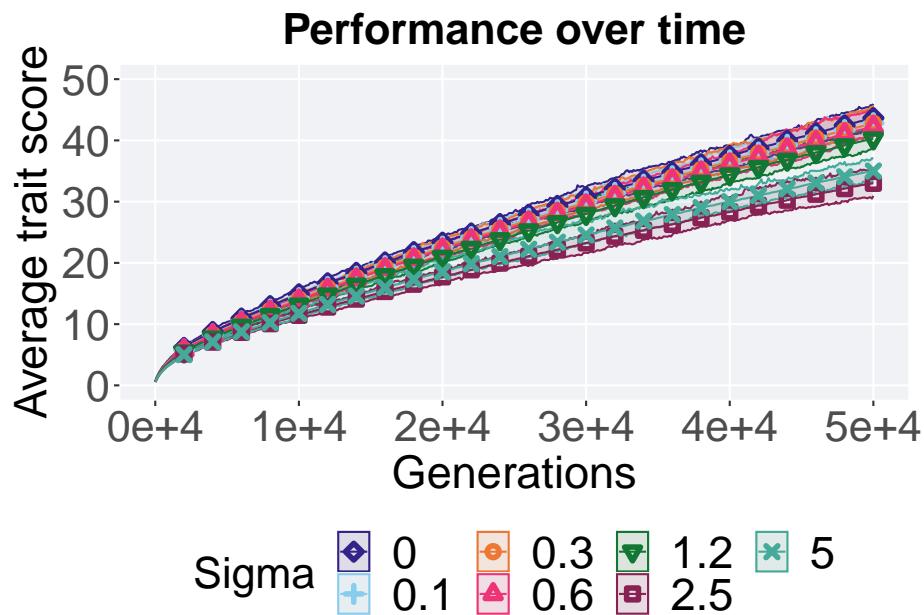
```
##  
##      0      0.1      0.3      0.6      1.2      2.5  
## 0.1 < 2e-16 -      -      -      -      -  
## 0.3 < 2e-16 1.5e-07 -      -      -      -  
## 0.6 < 2e-16 < 2e-16 3.0e-14 -      -      -  
## 1.2 < 2e-16 < 2e-16 < 2e-16 -      -  
## 2.5 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -  
## 5    < 2e-16 < 2e-16 < 2e-16 < 2e-16 1  
##  
## P value adjustment method: bonferroni
```

8.1.3 Multi-valley crossing

8.1.3.1 Performance over time

```
# data for lines and shading on plots  
lines = filter(gfs_ot_mvc, diagnostic == 'exploitation_rate') %>%  
  group_by(Sigma, 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 'Sigma'. You can override using the  
## `.groups` argument.  
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +  
  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="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) +
```

```
ggttitle('Performance over time')+
  p_theme +
  guides(
    shape=guide_legend(nrow=2, title.position = "left"),
    color=guide_legend(nrow=2, title.position = "left"),
    fill=guide_legend(nrow=2, title.position = "left")
  )
```



8.1.3.2 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(gfs_ot_mvc, diagnostic == 'exploitation_rate' & gen == 50000 & Sigma != 'ran')
end$Generation <- factor(end$gen)

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

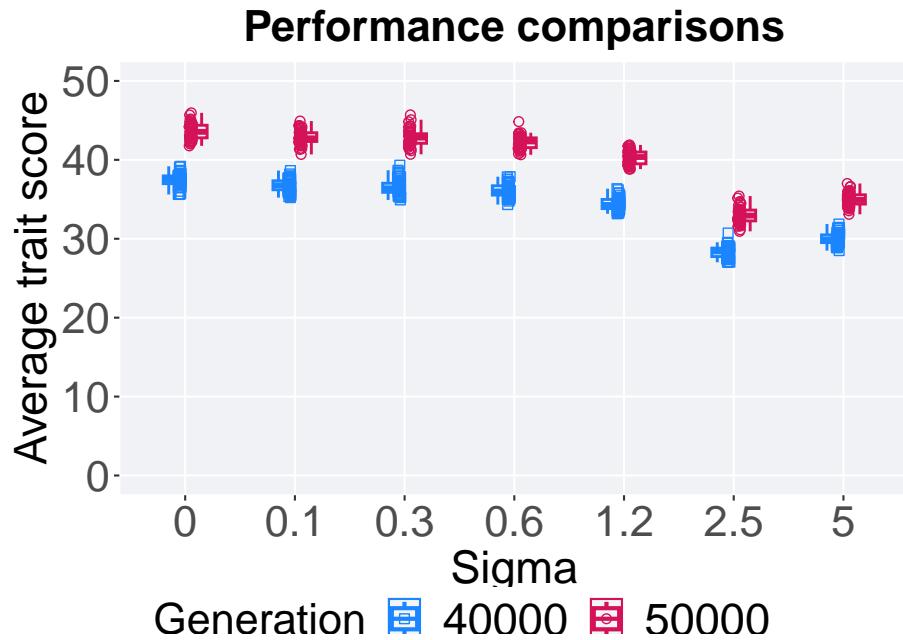
mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = Generation))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = 0.15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = 0.15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[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_discrete(
  name="Sigma"
) +
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
)

```



8.1.3.3 Stats

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

```
slices = filter(gfs_ot_mvc, diagnostic == 'exploitation_rate' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.
## # A tibble: 14 x 9
## # Groups:   Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <dbl> <dbl> <dbl> <dbl> <dbl>
```

```

## 1 0      50000      50      0 41.8   43.5   43.6   46.0 1.59
## 2 0      40000      50      0 35.6   37.5   37.5   39.2 0.977
## 3 0.1    50000      50      0 40.7   42.7   42.8   44.9 1.10
## 4 0.1    40000      50      0 35.2   36.8   36.7   38.6 1.14
## 5 0.3    50000      50      0 40.7   42.8   42.8   45.7 1.21
## 6 0.3    40000      50      0 34.9   36.4   36.6   39.3 1.15
## 7 0.6    50000      50      0 40.6   42.3   42.2   44.8 1.24
## 8 0.6    40000      50      0 34.3   36.0   36.1   37.9 1.18
## 9 1.2    50000      50      0 38.8   40.4   40.3   41.9 1.49
## 10 1.2   40000      50      0 33.2   34.3   34.4   36.3 1.16
## 11 2.5    50000      50      0 30.9   32.9   33.0   35.4 1.37
## 12 2.5   40000      50      0 27.0   28.4   28.3   30.7 1.07
## 13 5     50000      50      0 33.1   34.9   35.0   37.0 1.17
## 14 5     40000      50      0 28.5   29.9   30.0   31.9 0.977

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

```

```

## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 5.0

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction

```

```
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =  
## W = 2500, p-value < 2.2e-16  
## alternative hypothesis: true location shift is not equal to 0
```

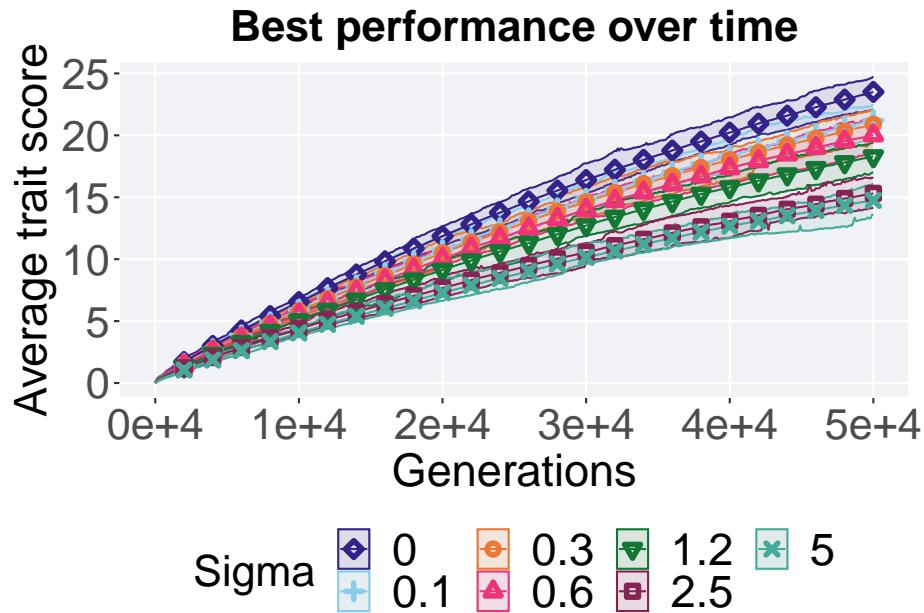
8.2 Ordered exploitation results

Here we present the results for **best performances** found by each genotypic fitness sharing sigma value 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 and 100.

8.2.1 Performance over time

Performance over time.

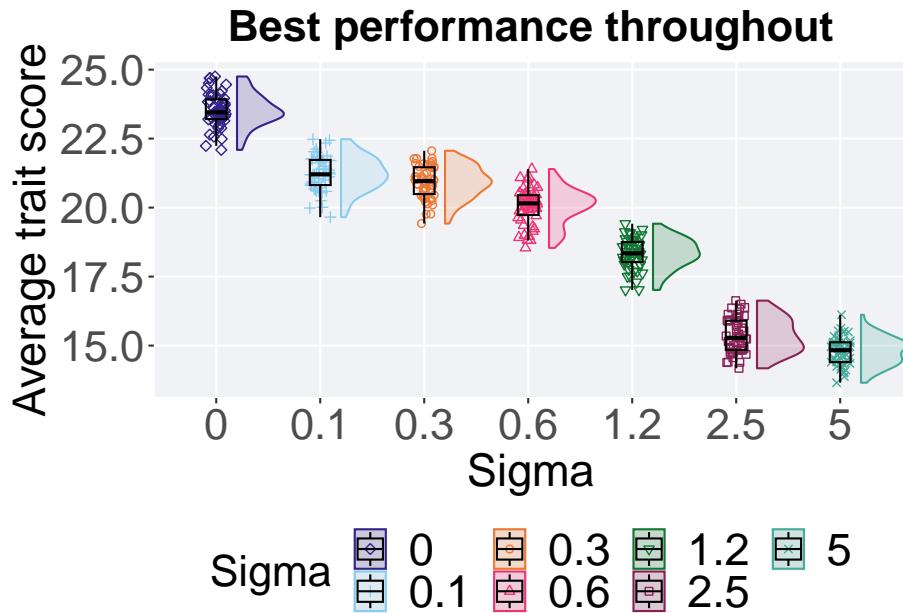
```
lines = filter(gfs_ot, diagnostic == 'ordered_exploitation') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(pop_fit_max),  
    mean = mean(pop_fit_max),  
    max = max(pop_fit_max)  
  )  
  
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## `.groups` argument.  
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma, shape =  
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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="Average trait score"  
  ) +  
  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("Best performance over time") +  
  p_theme
```



8.2.2 Best performance throughout

The best performance found throughout 50,000 generations.

```
filter(gfs_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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"
  ) +
  scale_x_discrete(
    name = "Sigma"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best performance throughout") +
  p_theme
```



8.2.2.1 Stats

Summary statistics about the best performance found.

```
performance = filter(gfs_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation')
group_by(performance, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt  min median  mean  max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 0      50      0  22.1  23.5  23.5  24.8  0.713
## 2 0.1    50      0  19.7  21.2  21.2  22.5  0.911
## 3 0.3    50      0  19.4  21.0  20.9  22.1  0.970
## 4 0.6    50      0  18.5  20.2  20.0  21.4  0.716
## 5 1.2    50      0  17.0  18.3  18.3  19.4  0.729
## 6 2.5    50      0  14.2  15.3  15.4  16.6  1.06 
## 7 5      50      0  13.7  14.8  14.8  16.1  0.717
```

Kruskal–Wallis test provides evidence of statistical differences for the best performance found in the population throughout 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 322.96, df = 6, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performance found in the population throughout 50,000 generations.

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

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
##          0         0.1        0.3        0.6        1.2        2.5
## 0.1 < 2e-16 -         -         -         -         -
## 0.3 < 2e-16 0.15850 -         -         -         -
## 0.6 < 2e-16 1.5e-11 5.6e-08 -         -         -
## 1.2 < 2e-16 < 2e-16 < 2e-16 3.4e-15 -         -
## 2.5 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
## 5 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 0.00038
##
## P value adjustment method: bonferroni
```

8.2.3 Multi-valley crossing

8.2.3.1 Performance comparison

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

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

mid = filter(gfs_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 40000)
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape =
geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), size = 100),
geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1]))
```

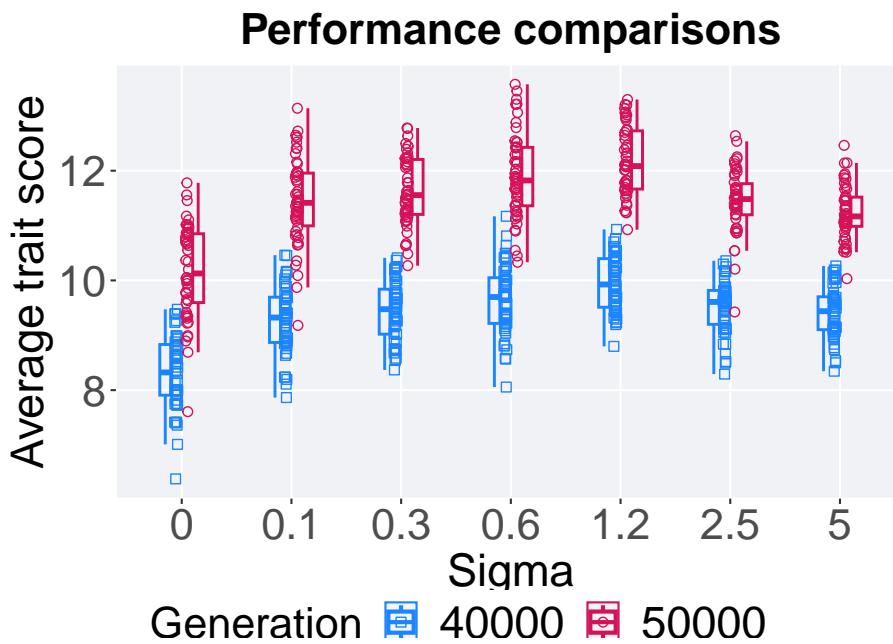
```

geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(y = 0))
geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(y = 0))

scale_y_continuous(
  name="Average trait score"
) +
scale_x_discrete(
  name="Sigma"
) +
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
)

```



8.2.3.1.1 Stats

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

```

slices = filter(gfs_ot_mvc, diagnostic == 'ordered_exploitation' & (gen == 50000 | gen
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## ` .groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int>  <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0      50000       50      0  7.61  10.1  10.2  11.8  1.26
## 2 0      40000       50      0  6.38  8.32  8.31  9.47  0.924
## 3 0.1    50000       50      0  9.18  11.4  11.5  13.1  0.962
## 4 0.1    40000       50      0  7.86  9.32  9.28  10.5  0.825
## 5 0.3    50000       50      0 10.3   11.6  11.6  12.8  1.00
## 6 0.3    40000       50      0  8.37  9.48  9.45  10.4  0.820
## 7 0.6    50000       50      0 10.3   11.8  11.9  13.6  1.07
## 8 0.6    40000       50      0  8.06  9.70  9.64  11.2  0.833
## 9 1.2    50000       50      0 10.9   12.1  12.2  13.3  1.06
## 10 1.2   40000       50      0  8.80  9.93  9.94  10.9  0.886
## 11 2.5   50000       50      0  9.42  11.5  11.4  12.6  0.568
## 12 2.5   40000       50      0  8.29  9.61  9.52  10.4  0.619
## 13 5     50000       50      0 10.0   11.2  11.2  12.5  0.533
## 14 5     40000       50      0  8.35  9.44  9.41  10.3  0.600

Sigma 0.0

wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

```

```
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2385, p-value = 5.239e-15
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2453, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2498, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6
wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2481, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2
wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

```

##  

## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices,  

## W = 2499, p-value < 2.2e-16  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 2.5  

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices,  

## W = 2464, p-value < 2.2e-16  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 5.0  

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices,  

## W = 2494, p-value < 2.2e-16  

## alternative hypothesis: true location shift is not equal to 0

```

8.3 Contraditory objectives diagnostic

Here we present the results for **satisfactory trait coverage** and **activation gene coverage** found by each genotypic fitness sharing sigma value replicate on the ordered exploitation diagnostic. Satisfactory trait coverage refers to the count of unique satisfied DIMENSIONALITY in the population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both coverage values fall between 0 and 100.

8.3.1 Satisfactory trait coverage

Satisfactory trait coverage analysis.

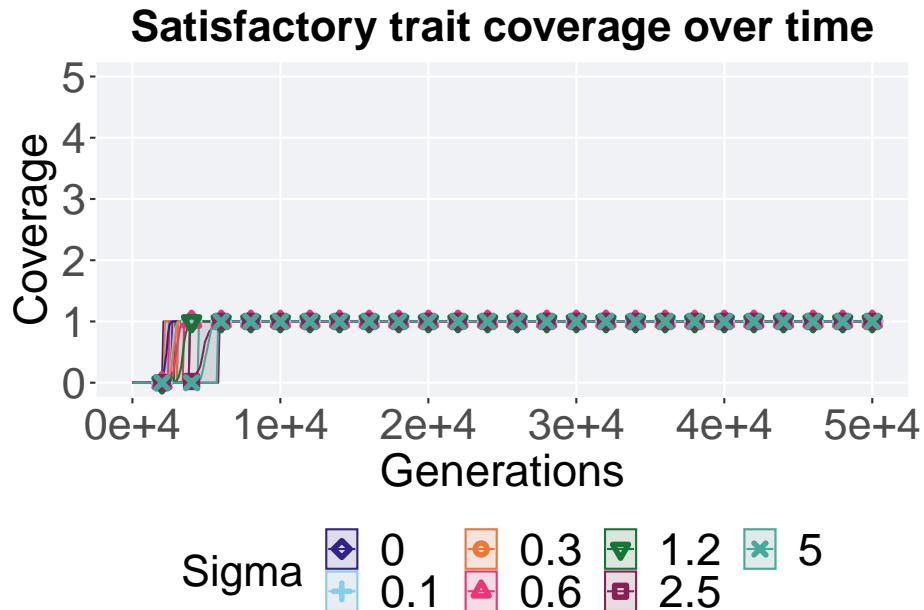
8.3.1.1 Coverage over time

Satisfactory trait coverage over time.

```
lines = filter(gfs_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

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

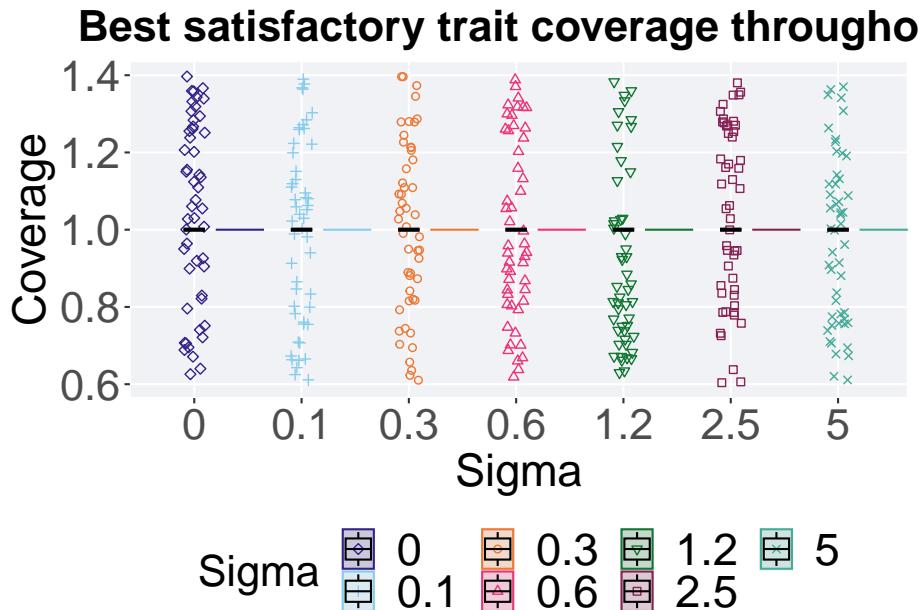
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage",
    limits=c(0, 5)
  ) +
  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
```



8.3.1.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
filter(gfs_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = Sigma, y = val, color = Sigma, fill = Sigma, shape = Sigma)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best satisfactory trait coverage throughout") +
  p_theme
```



8.3.1.2.1 Stats

Summary statistics for the best satisfactory trait coverage throughout 50,000 generations.

```
coverage = filter(gfs_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives')
  group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt  min median  mean  max  IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0      50     0     1     1     1     1     0
## 2 0.1    50     0     1     1     1     1     0
## 3 0.3    50     0     1     1     1     1     0
## 4 0.6    50     0     1     1     1     1     0
## 5 1.2    50     0     1     1     1     1     0
## 6 2.5    50     0     1     1     1     1     0
```

```
## 7 5      50      0      1      1      1      1      0
```

Kruskal–Wallis test provides evidence of no statistical difference among satisfactory trait coverage throughout 50,000 generations.

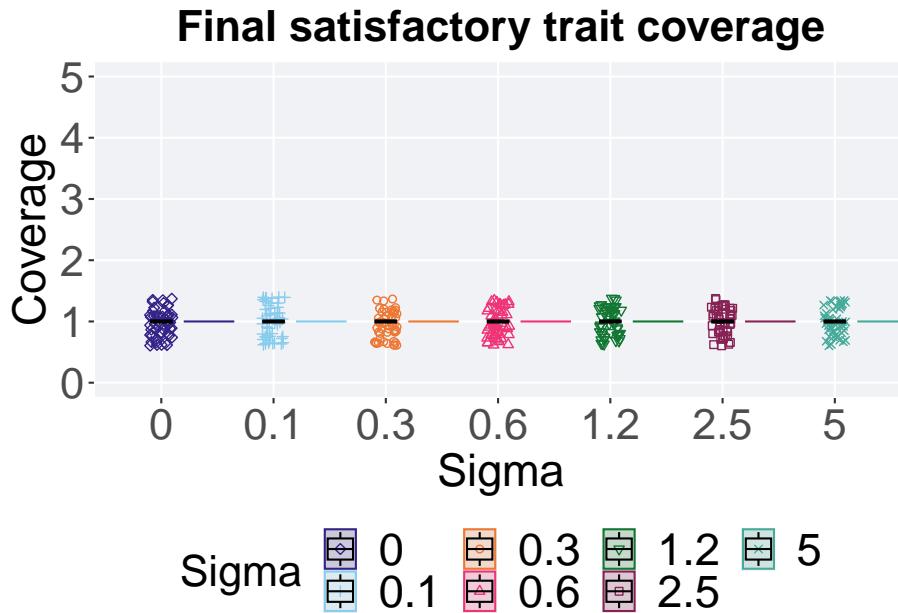
```
kruskal.test(val ~ Sigma, data = coverage)
```

```
##  
## Kruskal-Wallis rank sum test  
##  
## data: val by Sigma  
## Kruskal-Wallis chi-squared = NaN, df = 6, p-value = NA
```

8.3.1.3 End of 50,000 generations

Satisfactory trait coverage in the final population (50,000 generations).

```
filter(gfs_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%  
  ggplot(., aes(x = Sigma, y = pop_uni_obj, color = Sigma, fill = Sigma, shape = Sigma)) +  
    geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +  
    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, 5)  
    ) +  
    scale_x_discrete(  
      name = "Sigma"  
    ) +  
    scale_shape_manual(values = SHAPE) +  
    scale_colour_manual(values = cb_palette) +  
    scale_fill_manual(values = cb_palette) +  
    ggtitle("Final satisfactory trait coverage") +  
    p_theme
```



8.3.1.3.1 Stats

Summary statistics for satisfactory trait coverage in the final population (50,000 generations).

```
group_by(end, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 0      50     0     0     0     0     0     0     0
## 2 0.1    50     0     0     0     0.1    1     0     0
## 3 0.3    50     0     0     0     0.02   1     0     0
## 4 0.6    50     0     0     0     0.06   1     0     0
## 5 1.2    50     0     0     0     0.42   2     1     0
## 6 2.5    50     0     0     1     1     2     0     0
## 7 5      50     0     0     1     1.12  2     0     0
```

Kruskal–Wallis test provides evidence of no statistical difference among satisfactory trait coverage throughout 50,000 generations.

```
kruskal.test(pop_uni_obj ~ Sigma, data = end)

##
## Kruskal-Wallis rank sum test
##
## data: pop_uni_obj by Sigma
## Kruskal-Wallis chi-squared = 228.29, df = 6, p-value < 2.2e-16
```

8.3.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the contradictory objectives diagnostic.

8.3.2.1 Coverage over time

Activation gene coverage over time.

```
lines = filter(gfs_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

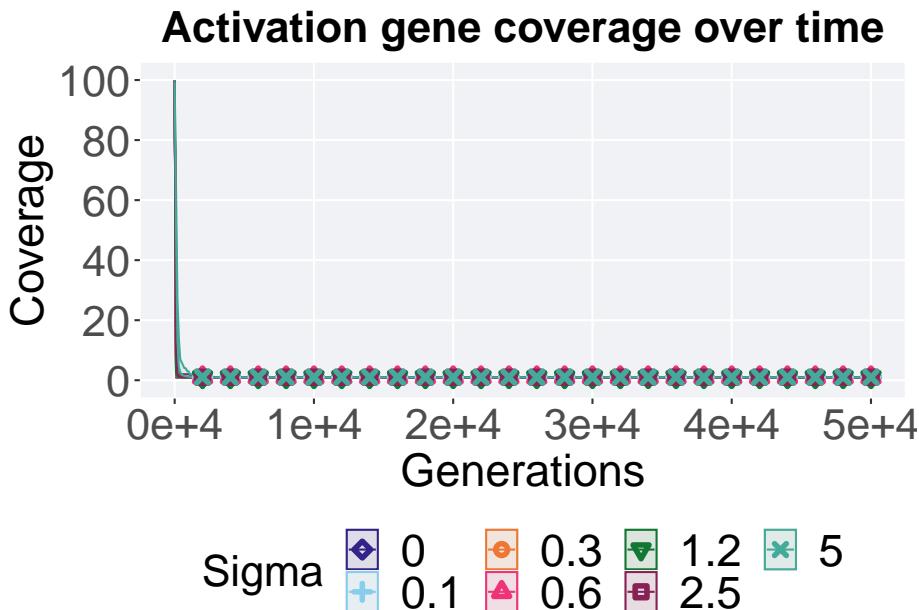
## `summarise()` has grouped output by 'Sigma'. You can override using the
## ` `.groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma))
  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(-1, 101),
    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

```



8.3.2.2 End of 50,000 generations

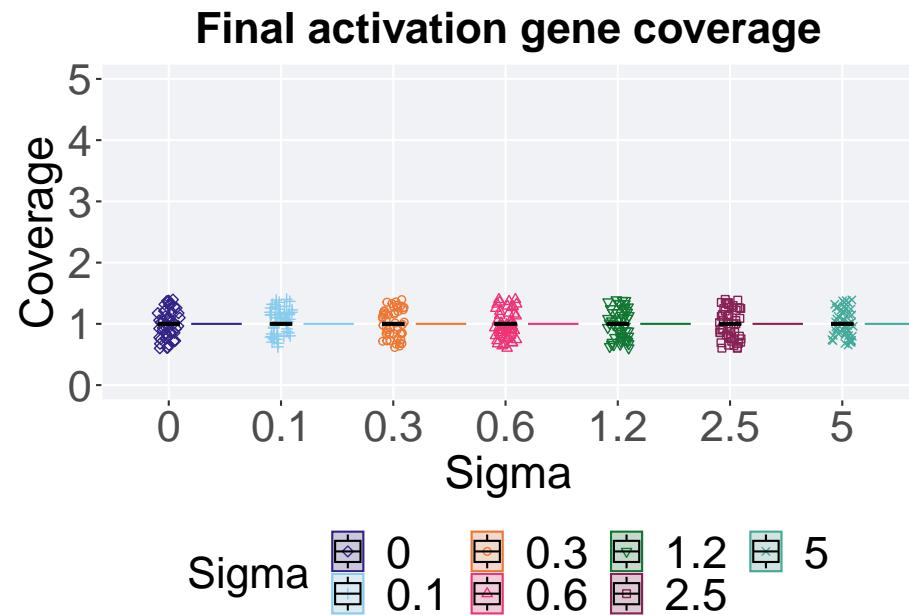
Activation gene coverage in the final population (50,000 generations).

```

filter(gfs_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = uni_str_pos, color = Sigma, fill = Sigma, shape = Sigma)) +
  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, 5)
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +

```

```
ggtile("Final activation gene coverage") +
  p_theme
```



8.3.2.2.1 Stats

Summary statistics for activation gene coverage in the final population (50,000 generations).

```
coverage = filter(gfs_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt  min median  mean  max  IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 0      50       0     1     1     1     1     0
## 2 0.1    50       0     1     1     1     1     0
## 3 0.3    50       0     1     1     1     1     0
```

```
## 4 0.6      50     0     1     1     1     1     0
## 5 1.2      50     0     1     1     1     1     0
## 6 2.5      50     0     1     1     1     1     0
## 7 5        50     0     1     1     1     1     0
```

Kruskal–Wallis test provides evidence of no statistical difference for activation gene coverage in the final population (50,000 generations).

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

```
##
## Kruskal-Wallis rank sum test
##
## data: uni_str_pos by Sigma
## Kruskal-Wallis chi-squared = NaN, df = 6, p-value = NA
```

8.3.3 Multi-valley crossing

8.3.3.1 Satisfactory trait coverage over time

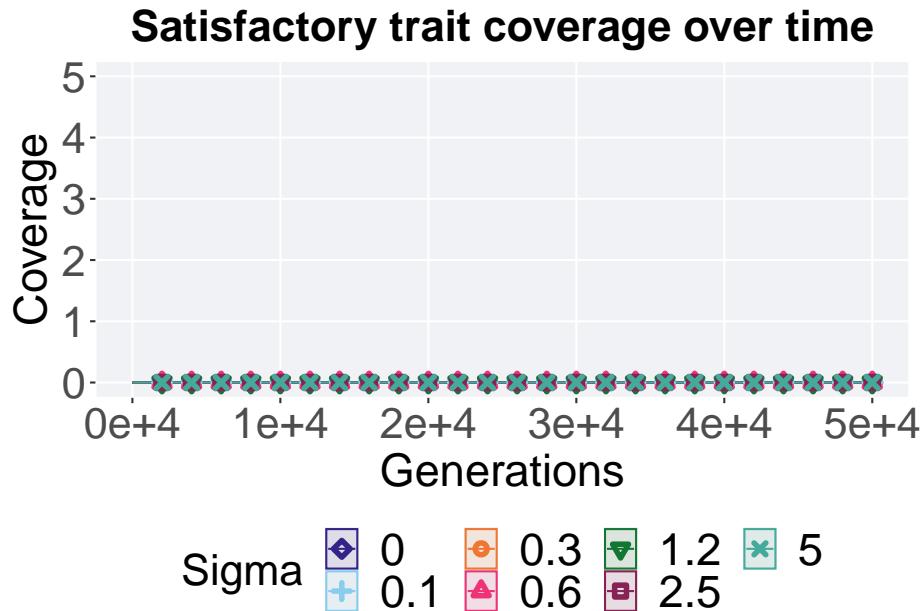
```
lines = filter(gfs_ot_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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)
  ) +
  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
```



8.3.3.2 Satisfactory trait coverage comparison

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

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

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

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

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

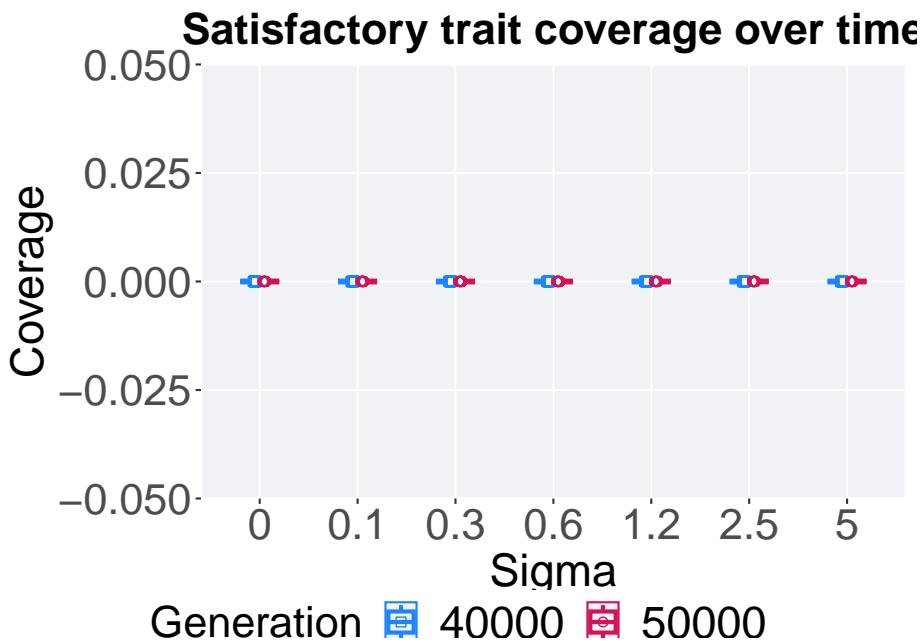
  scale_y_continuous(
    name="Coverage",
  ) +
```

```

scale_x_discrete(
  name="Sigma"
) +
  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 over time") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



8.3.3.2.1 Stats

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

```

slices = filter(gfs_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%

```

```

group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct>    <fct>     <int>   <int>   <int>   <dbl>   <dbl>   <int>   <dbl>
## 1 0        50000      50       0       0       0       0       0       0
## 2 0        40000      50       0       0       0       0       0       0
## 3 0.1      50000      50       0       0       0       0       0       0
## 4 0.1      40000      50       0       0       0       0       0       0
## 5 0.3      50000      50       0       0       0       0       0       0
## 6 0.3      40000      50       0       0       0       0       0       0
## 7 0.6      50000      50       0       0       0       0       0       0
## 8 0.6      40000      50       0       0       0       0       0       0
## 9 1.2      50000      50       0       0       0       0       0       0
## 10 1.2     40000      50       0       0       0       0       0       0
## 11 2.5     50000      50       0       0       0       0       0       0
## 12 2.5     40000      50       0       0       0       0       0       0
## 13 5       50000      50       0       0       0       0       0       0
## 14 5       40000      50       0       0       0       0       0       0

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1

```

```
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.1 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.3 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

Sigma 2.5

```
wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

## Wilcoxon rank sum test with continuity correction
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

Sigma 5.0

```
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

## Wilcoxon rank sum test with continuity correction
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 5 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
```

8.3.3.3 Activation gene coverage over time

```
lines = filter(gfs_ot_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

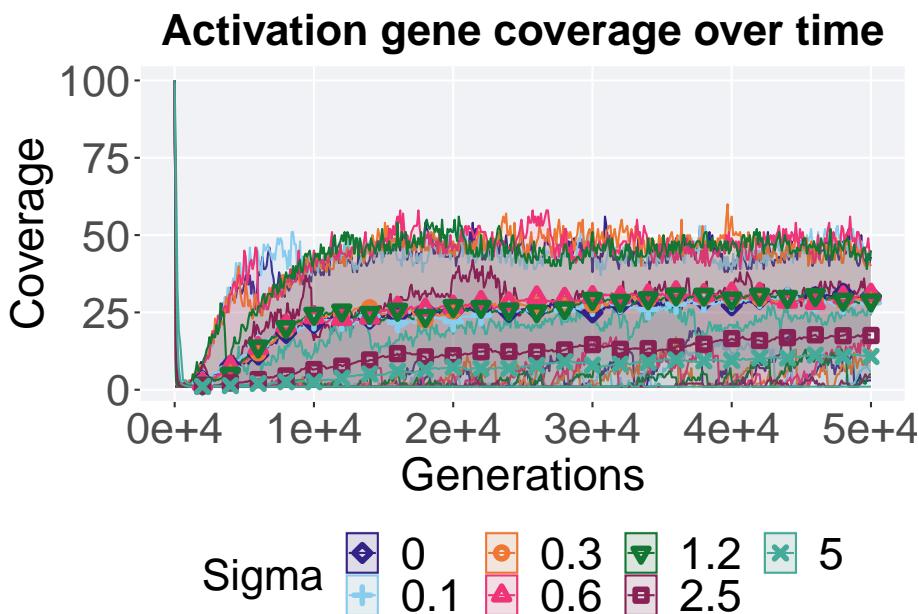
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2)
```

```

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

```



8.3.3.4 Activation gene coverage comparison

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

```

# 80% and final generation comparison
end = filter(gfs_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 50000)
end$Generation <- factor(end$gen)

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

mvc_p = ggplot(mid, aes(x = Sigma, y=uni_str_pos, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 10) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

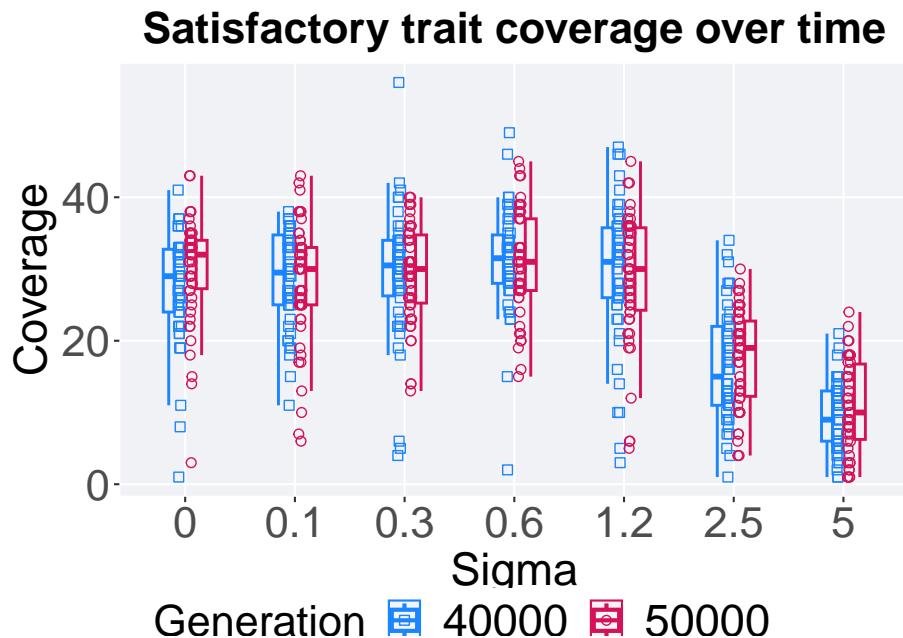
```

```

geom_point(data = end, aes(x = Sigma, y=uni_str_pos), col = mvc_col[2], position = position_nudge(x = 0.05)) +
  geom_boxplot(data = end, aes(x = Sigma, y=uni_str_pos), position = position_nudge(x = 0.05)) +
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  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 over time") +
  theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



8.3.3.4.1 Stats

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

```
slices = filter(gfs_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000, 40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <int> <dbl> <dbl> <int> <dbl>
## 1 0      50000       50     0     3    32  30.2    43   6.75
## 2 0      40000       50     0     1    29  27.3    41   8.75
## 3 0.1    50000       50     0     6    30  28.1    43   8
## 4 0.1    40000       50     0    11   29.5  28.7    38   9.75
## 5 0.3    50000       50     0    13   30  29.4    40   9.5
## 6 0.3    40000       50     0     4   30.5  29.3    56   7.75
## 7 0.6    50000       50     0    15   31   30.8    45   10
## 8 0.6    40000       50     0     2   31.5  31.2    49   6.75
## 9 1.2    50000       50     0     5   30   28.8    45   11.5
## 10 1.2   40000       50     0     3   31   29.4    47   9.75
## 11 2.5   50000       50     0     4   19   17.5    30   10.5
## 12 2.5   40000       50     0     1   15   16.5    34   11
## 13 5     50000       50     0     1   10   10.8    24   10.5
## 14 5     40000       50     0     1    9    9.5    21   7

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$uni_str_pos,
             alternative = 't')

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

```

## data: filter(slices, Sigma == 0 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1599, p-value = 0.01605
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1

wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.1 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1211, p-value = 0.7903
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1223.5, p-value = 0.8575
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1163, p-value = 0.5502
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```

##  

## data: filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  

## W = 1194, p-value = 0.7017  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 2.5  

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, Sigma == 2.5 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  

## W = 1380.5, p-value = 0.3696  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 5.0  

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, Sigma == 5.0 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  

## W = 1381.5, p-value = 0.3656  

## alternative hypothesis: true location shift is not equal to 0

```

8.4 Multi-path exploration results

Here we present the results for **best performances** and **activation gene coverage** found by each genotypic fitness sharing sigma value replicate on the multi-path exploration diagnostic. Best performance found refers to the largest average trait score found in a given population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both values fall between 0 and 100.

8.4.1 Performance

Here we analyze the performances for each parameter replicate on the multi-path exploration diagnostic.

8.4.1.1 Performance over time

Performance over time.

```

lines = filter(gfs_ot, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

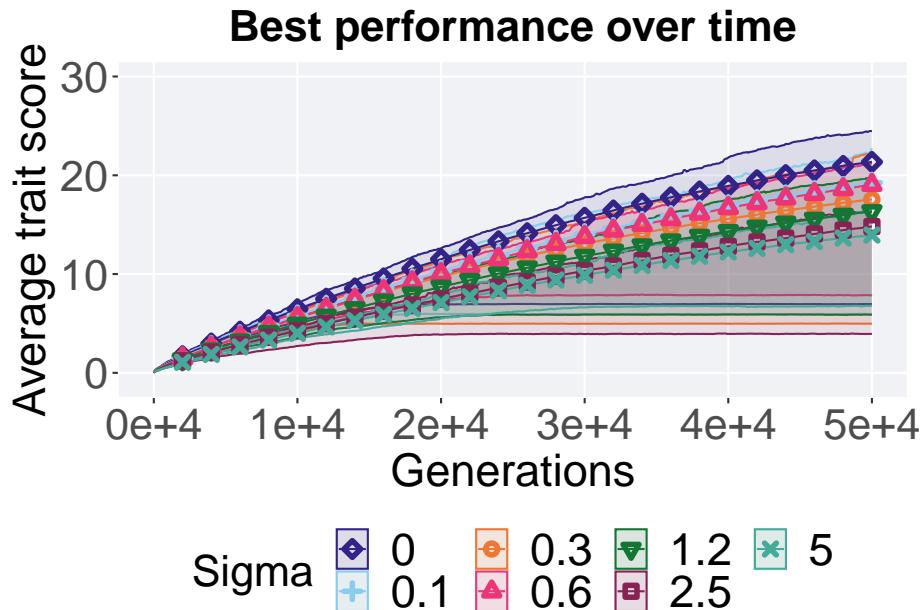
```

```

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

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color =
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2),
  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(-1, 30)
  ) +
  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("Best performance over time") +
  p_theme

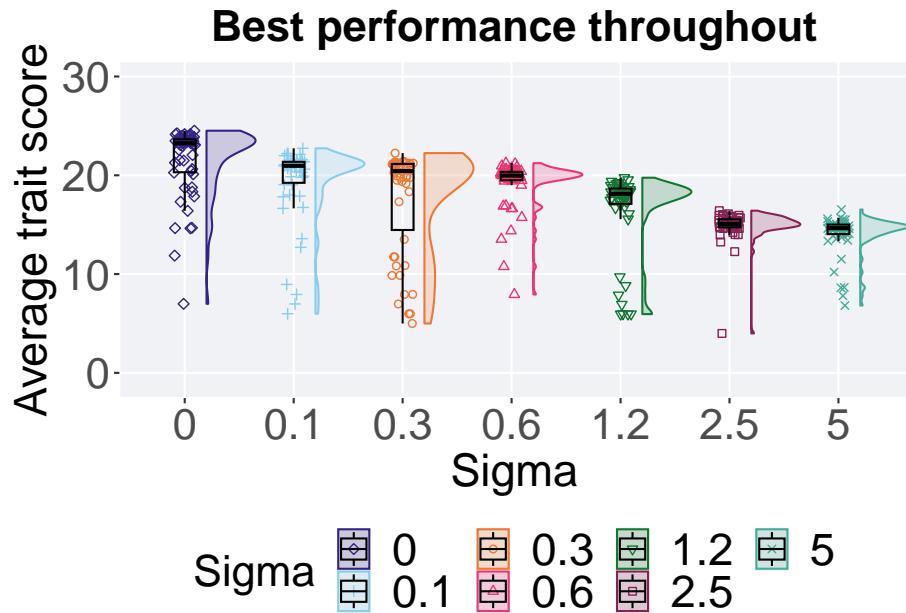
```



8.4.1.2 Best performance throughout

Here we plot the performance of the best performing solution found throughout 50,000 generations.

```
filter(gfs_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma))
  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, 30)
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best performance throughout") +
  p_theme
```



8.4.1.2.1 Stats

Summary statistics for the performance of the best performing solution.

```
performance = filter(gfs_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration')
group_by(performance, Sigma) %>%
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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0        50      0  6.99  23.2  21.4  24.5  3.34
## 2 0.1     50      0  5.98  21.0  19.3  22.7  2.12
## 3 0.3     50      0  4.99  20.4  17.6  22.2  6.69
## 4 0.6     50      0  7.93  20.0  19.1  21.2  0.841
## 5 1.2     50      0  5.95  18.1  16.4  19.8  1.55
## 6 2.5     50      0  3.99  15.1  14.9  16.4  0.809
## 7 5       50      0  6.81  14.6  14.0  16.5  0.962
```

Kruskal–Wallis test provides evidence of statistical differences among the best performing solution.

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

##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 161.29, df = 6, p-value < 2.2e-16

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

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

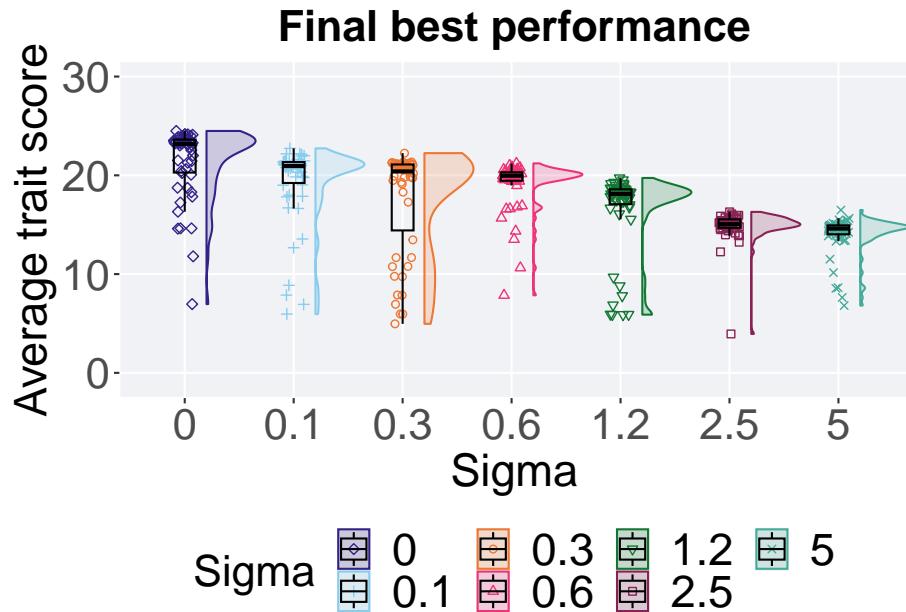
```
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
##      0      0.1     0.3     0.6     1.2     2.5
## 0.1 0.00026 -      -      -      -      -
## 0.3 4.8e-06 0.53186 -      -      -      -
## 0.6 8.5e-06 0.00531 1.00000 -      -      -
## 1.2 8.7e-09 2.1e-07 0.00023 2.7e-08 -      -
## 2.5 1.0e-11 4.1e-10 0.00024 4.2e-12 6.6e-08 -
## 5   9.4e-13 2.2e-10 9.9e-05 8.9e-13 3.7e-08 0.00268
##
## P value adjustment method: bonferroni
```

8.4.1.3 End of 50,000 generations

Best performance in the final population (50,000 generations).

```
filter(gfs_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = pop_fit_max / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma))
    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, 30)
    ) +
    scale_x_discrete(
      name = "Sigma"
    ) +
```

```
scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final best performance") +
  p_theme
```



8.4.1.3.1 Stats

Summary statistics for the best performance in the final population (50,000 generations).

```
performance = filter(gfs_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(performance, Sigma) %>%
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: 7 x 8
##   Sigma count na_cnt  min median  mean  max    IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
```

```
## 1 0      50     0  6.96  23.2  21.3  24.5 3.34
## 2 0.1    50     0  5.95  20.9  19.3  22.7 2.12
## 3 0.3    50     0  4.96  20.4  17.6  22.2 6.68
## 4 0.6    50     0  7.85  20.0  19.0  21.2 0.855
## 5 1.2    50     0  5.89  18.1  16.3  19.7 1.51
## 6 2.5    50     0  3.94  15.1  14.8  16.3 0.864
## 7 5      50     0  6.81  14.6  14.0  16.5 0.904
```

Kruskal–Wallis test provides evidence of statistical differences among best performance in the final population (50,000 generations).

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

```
##
## Kruskal-Wallis rank sum test
##
## data: pop_fit_max by Sigma
## Kruskal-Wallis chi-squared = 161.62, df = 6, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performance in the final population (50,000 generations).

```
pairwise.wilcox.test(x = performance$pop_fit_max, g = performance$Sigma , p.adjust.method = "bonf"
                      paired = FALSE, conf.int = FALSE, alternative = '1')
```

```
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$pop_fit_max and performance$Sigma
##
##      0      0.1     0.3     0.6     1.2     2.5
## 0.1 0.00024 -      -      -      -      -
## 0.3 5.0e-06 0.45941 -      -      -      -
## 0.6 8.2e-06 0.00431 1.00000 -      -      -
## 1.2 6.7e-09 1.9e-07 0.00022 2.9e-08 -      -
## 2.5 9.4e-12 4.1e-10 0.00024 4.4e-12 6.6e-08 -
## 5   9.9e-13 2.2e-10 9.6e-05 8.9e-13 3.6e-08 0.00283
##
## P value adjustment method: bonferroni
```

8.4.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the multi-path exploration diagnostic.

8.4.2.1 Coverage over time

Activation gene coverage over time.

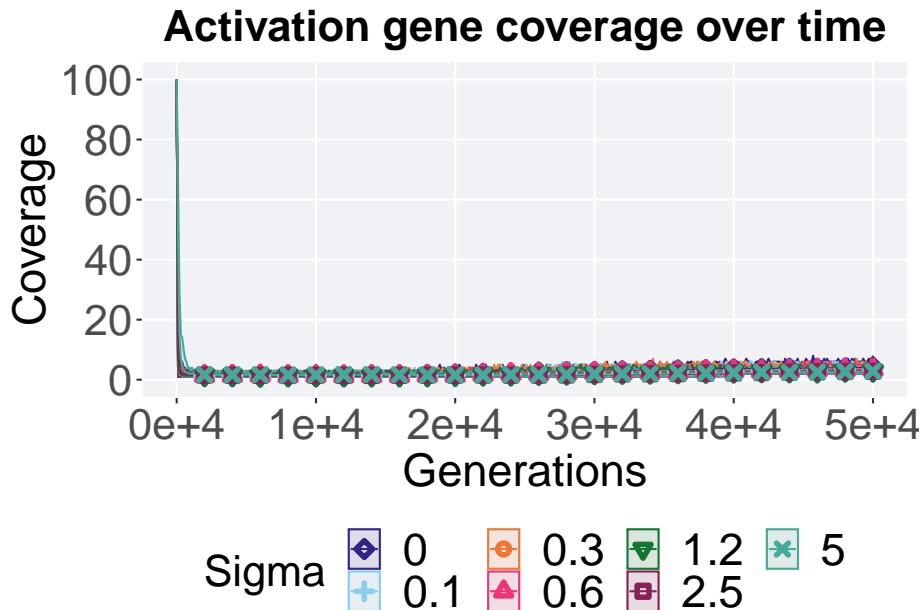
```

lines = filter(gfs_ot, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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(-1, 101),
    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

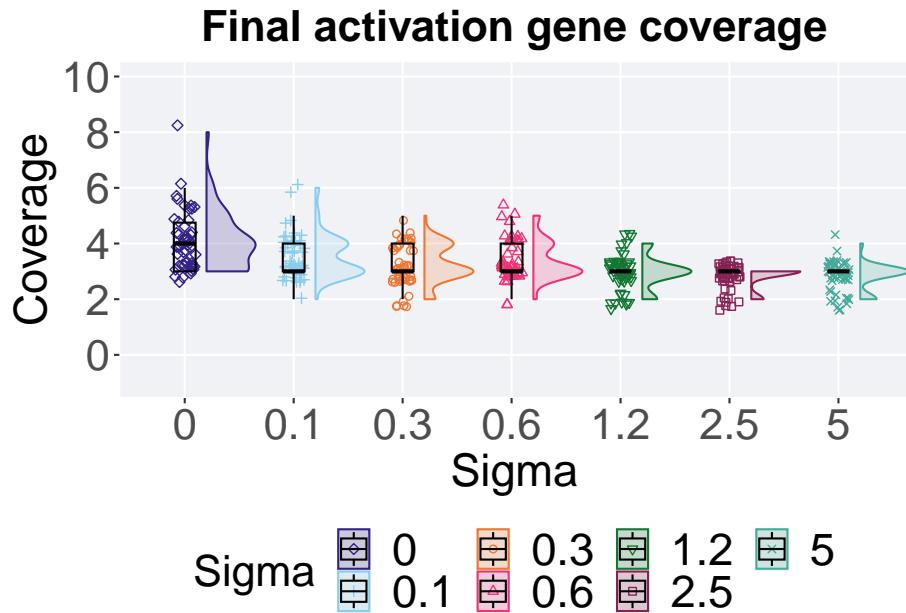
```



8.4.2.2 End of 50,000 generations

Activation gene coverage in the final population (50,000 generations).

```
filter(gfs_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = uni_str_pos, color = Sigma, fill = Sigma, shape = Sigma)) +
  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(-1, 10),
    breaks = seq(0, 10, 2),
    labels = c("0", "2", "4", "6", "8", "10")
  ) +
  scale_x_discrete(
    name = "Sigma"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final activation gene coverage") +
  p_theme
```



8.4.2.2.1 Stats

Summary statistics for the activation gene coverage in the final population (50,000 generations).

```
performance = filter(gfs_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(performance, Sigma) %>%
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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 0      50     0     0     3     4    4.04     8   1.75
## 2 0.1    50     0     0     2     3    3.54     6   1
## 3 0.3    50     0     0     2     3    3.24     5   1
## 4 0.6    50     0     0     2     3    3.38     5   1
## 5 1.2    50     0     0     2     3    2.94     4   0
## 6 2.5    50     0     0     2     3    2.8      3   0
```

```
## 7 5      50      0      2      3  2.8      4  0
```

Kruskal–Wallis test provides evidence of statistical differences among activation gene coverage in the final population (50,000 generations).

```
kruskal.test(uni_str_pos ~ Sigma, data = performance)
```

```
##  
## Kruskal-Wallis rank sum test  
##  
## data: uni_str_pos by Sigma  
## Kruskal-Wallis chi-squared = 93.885, df = 6, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the activation gene coverage in the final population (50,000 generations).

```
pairwise.wilcox.test(x = performance$uni_str_pos, g = performance$Sigma, p.adjust.method = "bonf"  
                      paired = FALSE, conf.int = FALSE, alternative = 't')
```

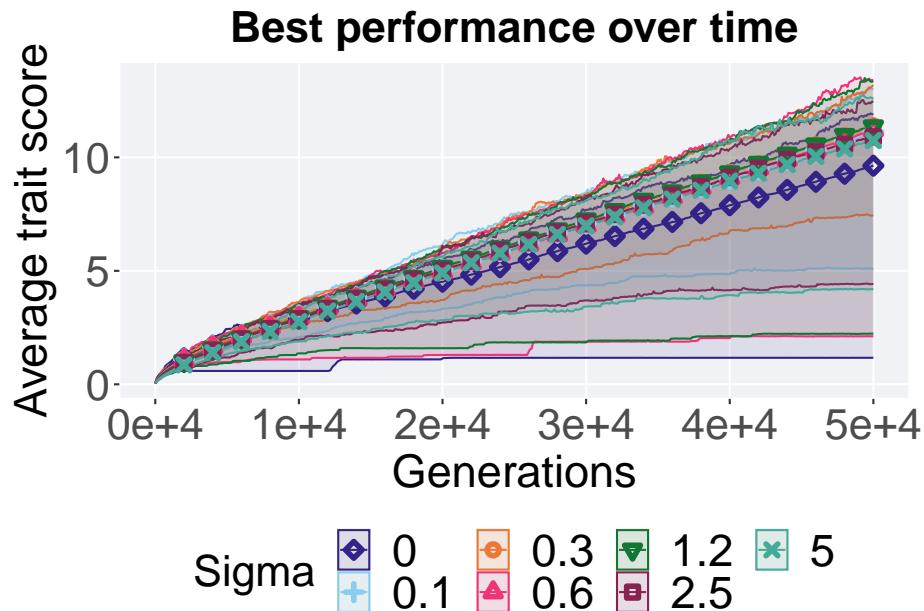
```
##  
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction  
##  
## data: performance$uni_str_pos and performance$Sigma  
##  
##      0      0.1      0.3      0.6      1.2      2.5  
## 0.1 0.16972 -      -      -      -      -  
## 0.3 0.00109 1.00000 -      -      -      -  
## 0.6 0.00962 1.00000 1.00000 -      -      -  
## 1.2 1.2e-07 0.00138 0.48268 0.03870 -      -  
## 2.5 5.9e-11 7.7e-07 0.00563 6.8e-05 1.00000 -  
## 5   4.4e-10 4.4e-06 0.01196 0.00025 1.00000 1.00000  
##  
## P value adjustment method: bonferroni
```

8.4.3 Multi-valley crossing

8.4.3.1 Performance over time

```
lines = filter(gfs_ot_mvc, diagnostic == 'multipath_exploration') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(pop_fit_max),  
    mean = mean(pop_fit_max),  
    max = max(pop_fit_max)  
  )  
  
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## ` .groups` argument.
```

```
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma)) +
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2) +
  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"
  ) +
  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("Best performance over time") +
  p_theme
```



8.4.3.2 Performance comparisons

```
# 80% and final generation comparison
end = filter(gfs_ot_mvc, diagnostic == 'multipath_exploration' & gen == 50000)
```

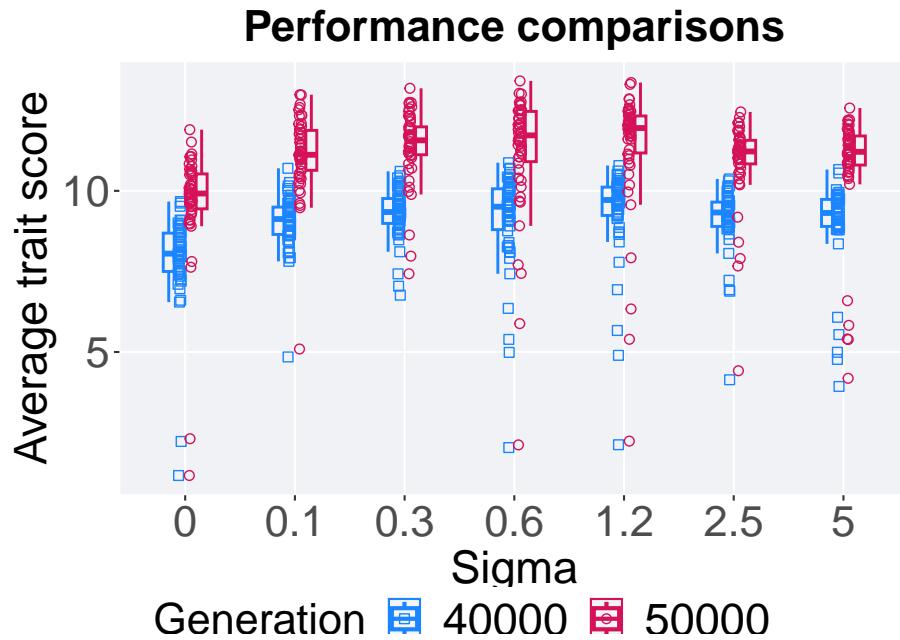
```
end$Generation <- factor(end$gen)

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

mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = Generation))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 100)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])
  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = 0.15, y = 0), size = 100)
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = 0.15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  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
)
```



8.4.3.2.1 Stats

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

```
# 80% and final generation comparison
slices = filter(gfs_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000, 40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.

## # A tibble: 14 x 9
## # Groups:   Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max     IQR
##   <dbl>      <dbl>   <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

##   <fct> <fct>    <int>  <int> <dbl>  <dbl> <dbl> <dbl>
## 1 0      50000       50     0  1.17  9.92  9.63 11.9  1.08
## 2 0      40000       50     0  1.17  8.05  7.89  9.67 11.19
## 3 0.1    50000       50     0  5.09 11.1  11.1  13.0  1.24
## 4 0.1    40000       50     0  4.85  9.13  9.05 10.7  0.848
## 5 0.3    50000       50     0  7.43 11.6  11.4  13.2  0.865
## 6 0.3    40000       50     0  6.76  9.34  9.32 10.6  0.772
## 7 0.6    50000       50     0  2.12 11.7  11.2  13.4  1.56
## 8 0.6    40000       50     0  2.04  9.51  9.10 10.9  1.27
## 9 1.2    50000       50     0  2.24 11.9  11.4  13.4  1.15
## 10 1.2   40000       50     0  2.12  9.72  9.33 10.8  0.880
## 11 2.5    50000       50     0  4.42 11.2  10.9  12.4  0.732
## 12 2.5    40000       50     0  4.14  9.33  9.12 10.4  0.754
## 13 5     50000       50     0  4.18 11.2  10.7  12.6  0.902
## 14 5     40000       50     0  3.93  9.31  8.96 10.7  0.838

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2284, p-value = 1.043e-12
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2397, p-value = 2.706e-15
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```

##  

## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices,  

## W = 2327, p-value = 1.161e-13  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 0.6  

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices,  

## W = 2197, p-value = 6.8e-11  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 1.2  

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices,  

## W = 2250, p-value = 5.565e-12  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 2.5  

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices,  

## W = 2281, p-value = 1.211e-12  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 5.0  

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

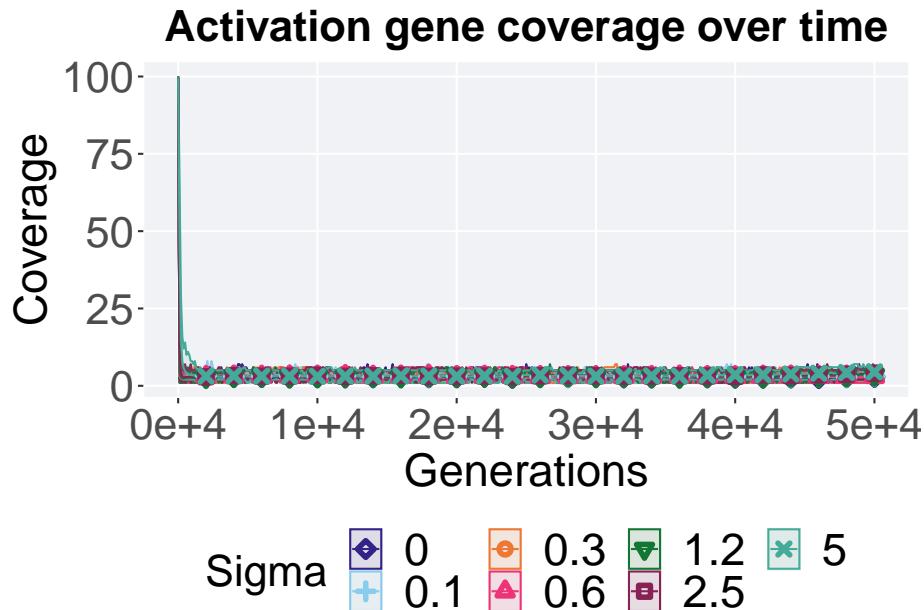
##
```

```
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2256, p-value = 4.157e-12
## alternative hypothesis: true location shift is not equal to 0
```

```
lines = filter(gfs_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )
```

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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"
  ) +
  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
```



8.4.3.4 Activation gene coverage comparison

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

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

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

mvc_p = ggplot(mid, aes(x = Sigma, y=uni_str_pos, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1], position = position_jitternudge(jitter.width = .03, nudge_x = -.15), size = 1.5) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=uni_str_pos), col = mvc_col[2], position = position_nudge(x = 0, y = 0)) +
  geom_boxplot(data = end, aes(x = Sigma, y=uni_str_pos), position = position_nudge(x = 0, y = 0), lwd = 0.7, col = mvc_col[2])

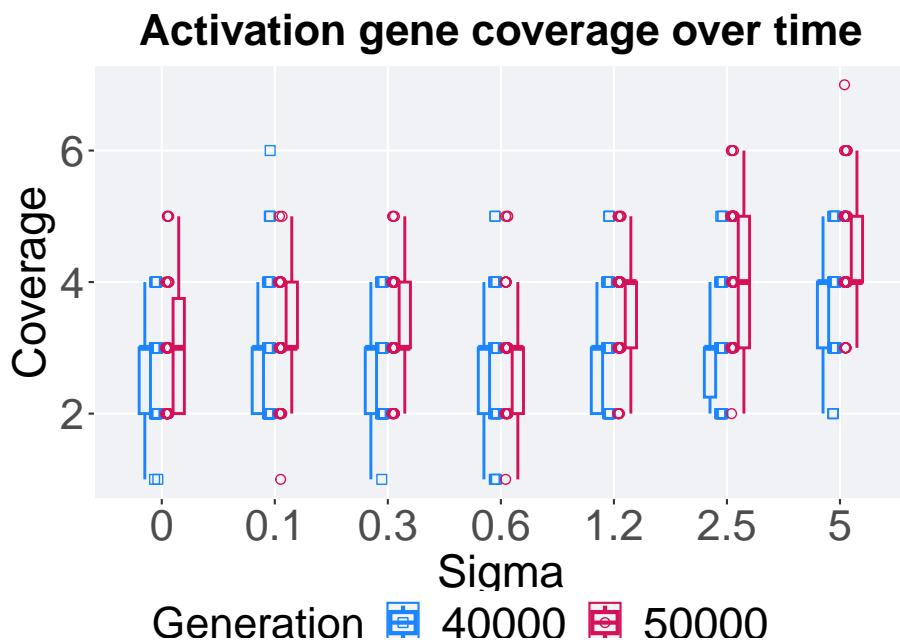
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=c(0,1)) +
```

```

scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
p_theme

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

```



8.4.3.4.1 Stats

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

```

slices = filter(gfs_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(uni_str_pos)),
    mean_coverage = mean(coverage),
    median_coverage = median(coverage),
    min_coverage = min(coverage),
    max_coverage = max(coverage)
  )

```

```

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 'Sigma'. You can override using the
## `.groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct>    <fct>     <int>   <int>   <dbl>   <dbl>   <int>   <dbl>
## 1 0        50000      50       0     2     3    2.86     5   1.75
## 2 0        40000      50       0     1     3    2.62     4   1
## 3 0.1      50000      50       0     1     3    3.1      5   1
## 4 0.1      40000      50       0     2     3    2.86     6   1
## 5 0.3      50000      50       0     2     3    3.14     5   1
## 6 0.3      40000      50       0     1     3    2.72     4   1
## 7 0.6      50000      50       0     1     3    2.92     5   1
## 8 0.6      40000      50       0     1     3    2.88     5   1
## 9 1.2      50000      50       0     2     4    3.5      5   1
## 10 1.2     40000      50       0     2     3    3         5   1
## 11 2.5     50000      50       0     2     4    4.1      6   2
## 12 2.5     40000      50       0     2     3    3.04     5   0.75
## 13 5       50000      50       0     3     4    4.48     7   1
## 14 5       40000      50       0     2     4    3.6      5   1

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1366, p-value = 0.3922
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##

```

```
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos and filter(slices, Sigma == 0.1 & Generation == 40000)$uni_str_pos
## W = 1498.5, p-value = 0.07098
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos and filter(slices, Sigma == 0.3 & Generation == 40000)$uni_str_pos
## W = 1573, p-value = 0.01769
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos and filter(slices, Sigma == 0.6 & Generation == 40000)$uni_str_pos
## W = 1245.5, p-value = 0.9763
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos and filter(slices, Sigma == 1.2 & Generation == 40000)$uni_str_pos
## W = 1661.5, p-value = 0.002736
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$uni_str_pos,
            alternative = 't')
```

```
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos and filter(slices,  
## W = 1948, p-value = 4.828e-07  
## alternative hypothesis: true location shift is not equal to 0  
Sigma 5.0  
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$uni_str_pos,  
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$uni_str_pos,  
            alternative = 't')  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$uni_str_pos and filter(slices,  
## W = 1874.5, p-value = 4.82e-06  
## alternative hypothesis: true location shift is not equal to 0
```

Chapter 9

Phenotypic fitness sharing

We present the results from our parameter sweep on phenotypic fitness sharing. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, performance with no similarity penalty and stochastic remainder selection is used to identify parent solutions.

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

9.1 Exploitation rate results

Here we present the results for **best performances** found by each phenotypic fitness sharing sigma value replicate on the exploitation rate diagnostic. Best performance found refers to the largest average trait score found in a given population. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, no similarity penalty is used, and only stochastic remainder selection is used to identify parent solutions.

9.1.1 Performance over time

Performance over time.

```
lines = filter(pfs_ot, diagnostic == 'exploitation_rate') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
```

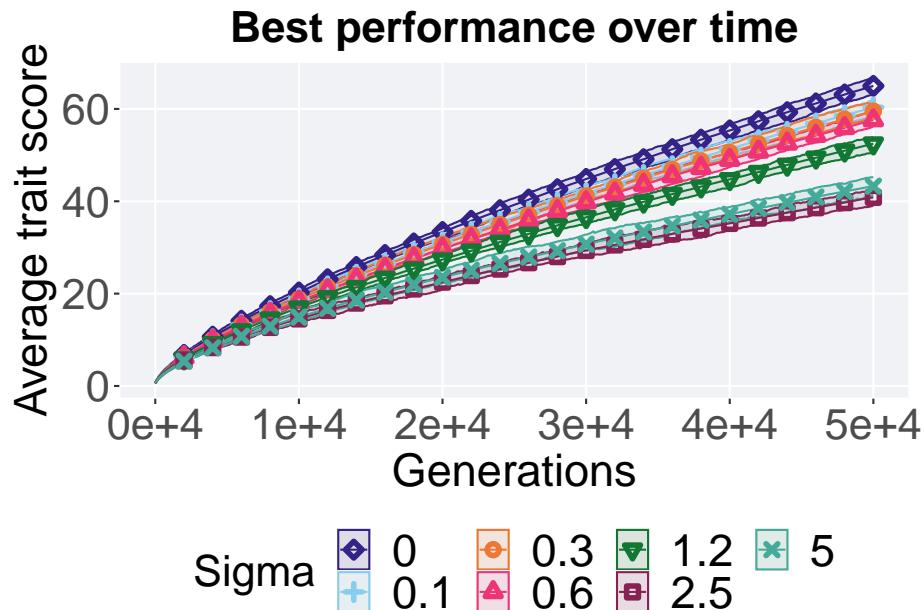
```

    max = max(pop_fit_max)
)

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

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color =
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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"
  ) +
  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("Best performance over time") +
  p_theme

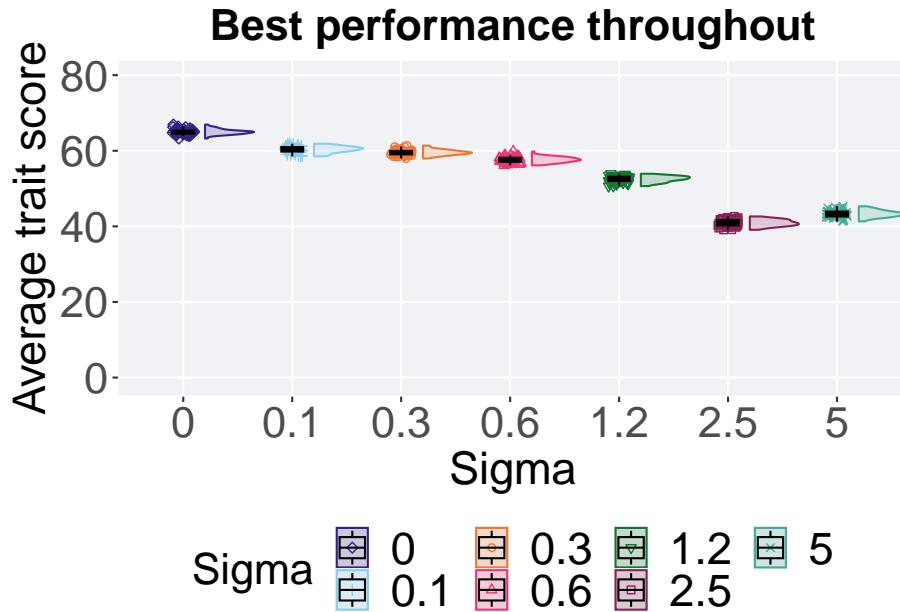
```



9.1.2 Best performance throughout

The best performance found throughout 50,000 generations.

```
filter(pfs_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
    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, 80),
      breaks=seq(0,80, 20),
      labels=c("0", "20", "40", "60", "80")
    ) +
    scale_x_discrete(
      name="Sigma"
    ) +
    scale_shape_manual(values=SHAPE) +
    scale_colour_manual(values = cb_palette) +
    scale_fill_manual(values = cb_palette) +
    ggtitle("Best performance throughout") +
    p_theme
```



9.1.2.1 Stats

Summary statistics for the best performance found.

```
performance = filter(pfs_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate'
group_by(performance, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int>   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0         50     0  63.2   65.0   65.0  66.9  0.816
## 2 0.1       50     0  58.5   60.4   60.4  61.9  1.19
## 3 0.3       50     0  58.0   59.5   59.5  61.4  0.908
## 4 0.6       50     0  56.2   57.6   57.6  59.8  1.11
## 5 1.2       50     0  50.7   52.6   52.5  53.9  1.11
## 6 2.5       50     0  39.1   40.9   40.9  42.6  1.40
## 7 5         50     0  41.3   43.3   43.3  45.3  1.29
```

Kruskal–Wallis test provides evidence of significant differences among sigma values on the best performance found.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 335.72, df = 6, p-value < 2.2e-16
```

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

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

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
```

```

##      0      0.1      0.3      0.6      1.2      2.5
## 0.1 < 2e-16 -      -      -      -      -
## 0.3 < 2e-16 2.5e-05 -      -      -      -
## 0.6 < 2e-16 3.3e-16 7.4e-15 -      -      -
## 1.2 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -      -
## 2.5 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
## 5    < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 1
##
## P value adjustment method: bonferroni

```

9.1.3 Multi-valley crossing

9.1.3.1 Performance comparison

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

```

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

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

mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = Generation))
  geom_point(col = mvc_col[1], position = position_jitternudge(jitter.width = .03, nudge.x = -.05))
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = -.15, y = 0))
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[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_discrete(
  name="Scheme"
) +
scale_shape_manual(values=c(0,1)) +
scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
p_theme

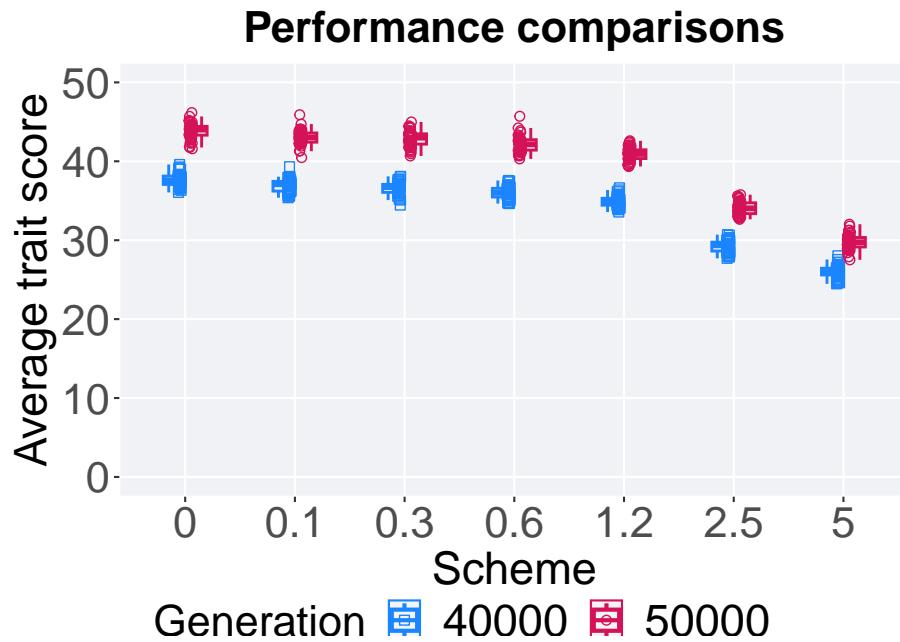
plot_grid(
  mvc_p +
  ggtitle("Performance comparisons") +
  theme(legend.position="none"),

```

```

legend,
nrow=2,
rel_heights = c(1,.05),
label_size = TSIZE
)

```



9.1.3.2 Stats

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

```

slices = filter(pfs_ot_mvc, diagnostic == 'exploitation_rate' & (gen == 50000 | gen ==
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the

```

```

## `^.groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0      50000      50     0  41.6  44.0  43.8  46.2 1.11
## 2 0      40000      50     0  36.1  37.6  37.6  39.6 1.05
## 3 0.1    50000      50     0  40.5  43.0  43.0  45.9 1.16
## 4 0.1    40000      50     0  35.4  37.1  36.9  39.3 1.23
## 5 0.3    50000      50     0  40.7  43.0  42.8  45.0 1.30
## 6 0.3    40000      50     0  34.4  36.7  36.6  38.1 1.01
## 7 0.6    50000      50     0  40.3  42.2  42.1  45.7 1.32
## 8 0.6    40000      50     0  34.6  36.0  36.0  37.6 1.10
## 9 1.2    50000      50     0  39.3  40.9  40.9  42.6 1.11
## 10 1.2   40000      50     0  33.6  34.9  34.9  36.6 0.875
## 11 2.5   50000      50     0  32.6  34.0  34.1  35.8 1.37
## 12 2.5   40000      50     0  27.7  29.3  29.2  30.7 1.12
## 13 5     50000      50     0  27.5  29.7  29.7  32.0 1.20
## 14 5     40000      50     0  24.5  26.0  26.0  28.0 0.844

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma ==
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

```

```
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

Sigma 5.0

```
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2497, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

9.2 Ordered exploitation results

Here we present the results for **best performances** found by each phenotypic fitness sharing sigma value replicate on the ordered exploitation diagnostic. Best performance found refers to the largest average trait score found in a given population. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, no similarity penalty is used, and only stochastic remainder selection is used to identify parent solutions.

9.2.1 Performance over time

Performance over time.

```
lines = filter(pfs_ot, diagnostic == 'ordered_exploitation') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

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

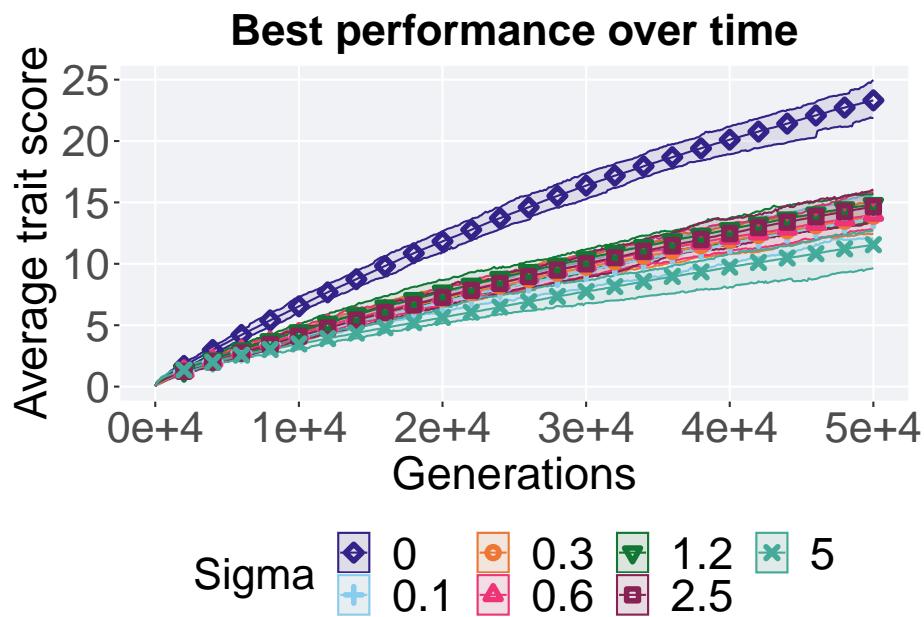
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma, size = 1.5, alpha = 0.1)) +
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha = 1) +
  scale_y_continuous(
    name="Average trait score"
  ) +
  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("Best performance over time") +
p_theme

```



9.2.2 Best performance throughout

The best performance found throughout 50,000 generations.

```

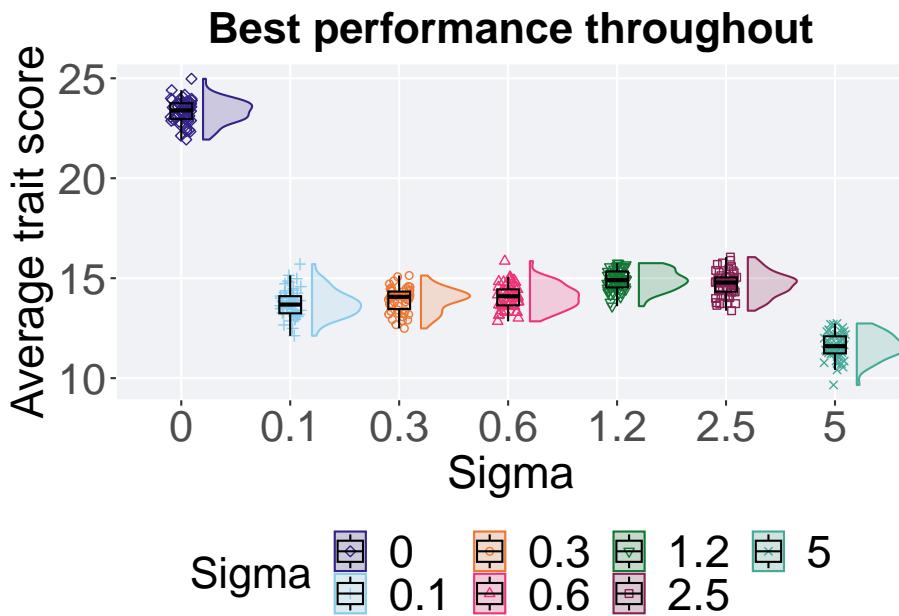
filter(pfs_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  geom_point(position = position_jitter(width = .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"
  ) +
  scale_x_discrete(
    name = "Sigma"
  )

```

```

scale_shape_manual(values=SHAPE) +
scale_colour_manual(values = cb_palette) +
scale_fill_manual(values = cb_palette) +
ggtitle("Best performance throughout") +
p_theme

```



9.2.2.1 Stats

Summary statistics about the best performance found.

```

performance = filter(pfs_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation')
group_by(performance, Sigma) %>%
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: 7 x 8
##   Sigma count  na_cnt  min  median  mean  max    IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     0     25      0  22.0  23.0  23.0  25.0  2.0
## 2     0.1    14      0  12.0  14.0  14.0  15.0  2.0
## 3     0.3    14      0  13.0  14.0  14.0  16.0  2.0
## 4     0.6    14      0  13.0  14.0  14.0  16.0  2.0
## 5     1.2    14      0  14.0  14.0  14.0  15.0  1.0
## 6     2.5    14      0  14.0  15.0  15.0  16.0  1.0
## 7     5      12      0  10.0  12.0  12.0  12.0  2.0

```

```
## 1 0      50    0 21.9    23.4 23.3 25.0 0.785
## 2 0.1    50    0 12.1    13.7 13.7 15.7 0.858
## 3 0.3    50    0 12.5    14.1 13.9 15.1 0.871
## 4 0.6    50    0 12.8    14.1 14.1 15.9 0.801
## 5 1.2    50    0 13.6    14.9 14.9 15.8 0.801
## 6 2.5    50    0 13.4    14.8 14.7 16.1 0.713
## 7 5      50    0  9.66   11.6 11.6 12.7 0.854
```

Kruskal–Wallis test provides evidence of statistical differences for the best performance found in the population throughout 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 265.26, df = 6, p-value < 2.2e-16

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performance found in the population throughout 50,000 generations.

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

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
##      0     0.1    0.3    0.6    1.2    2.5
## 0.1 <2e-16 -     -     -     -     -
## 0.3 <2e-16 1.00 -     -     -     -
## 0.6 <2e-16 1.00 1.00 -     -     -
## 1.2 <2e-16 1.00 1.00 1.00 -     -
## 2.5 <2e-16 1.00 1.00 1.00 0.76 -
## 5   <2e-16 <2e-16 <2e-16 <2e-16 <2e-16 <2e-16

##
## P value adjustment method: bonferroni
```

9.2.3 Multi-valley crossing

9.2.3.1 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(pfs_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 50000)
end$Generation <- factor(end$gen)

mid = filter(pfs_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 40000)
mid$Generation <- factor(mid$gen)

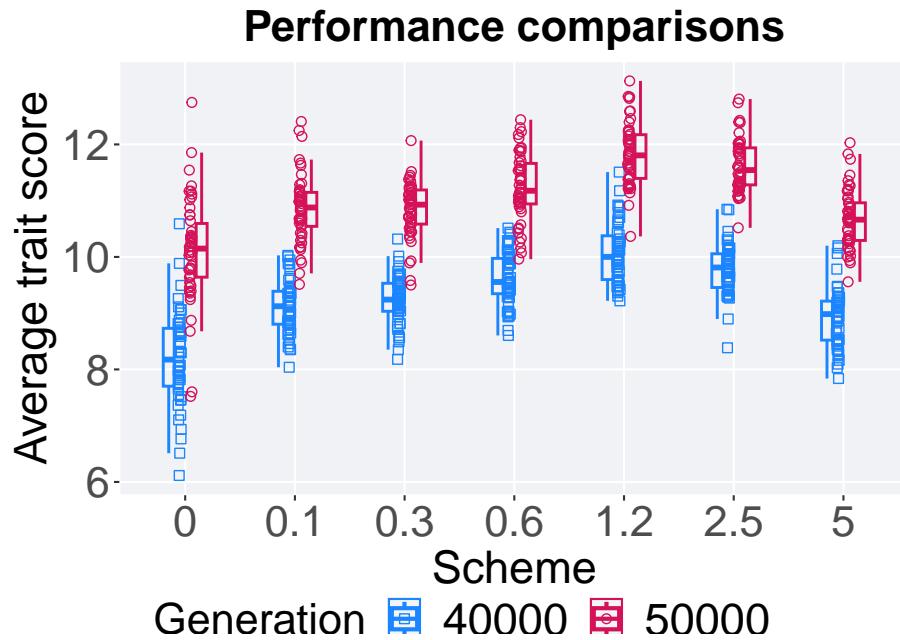
mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = Generation))
  geom_point(col = mvc_col[1], position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), nudge.y = 0)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = 0.15, y = 0))
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = 0.15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  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
)

```



9.2.3.2 Stats

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

```
slices = filter(pfs_ot_mvc, diagnostic == 'ordered_exploitation' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.
## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max     IQR
##   <fct> <fct>     <int>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
```

```

## 1 0      50000      50      0  7.52  10.1  10.1   12.7 0.955
## 2 0      40000      50      0  6.11  8.18  8.19   10.6 1.03
## 3 0.1    50000      50      0  9.51  10.9  10.9   12.4 0.604
## 4 0.1    40000      50      0  8.04  9.12  9.12   10.0 0.586
## 5 0.3    50000      50      0  9.50  10.9  10.8   12.1 0.606
## 6 0.3    40000      50      0  8.18  9.24  9.23   10.3 0.498
## 7 0.6    50000      50      0  9.96  11.2  11.2   12.4 0.721
## 8 0.6    40000      50      0  8.60  9.55  9.62   10.5 0.628
## 9 1.2    50000      50      0 10.4   11.8  11.8   13.1 0.777
## 10 1.2   40000      50      0  9.22  10.0  10.1   11.5 0.778
## 11 2.5    50000      50      0 10.5   11.5  11.6   12.8 0.658
## 12 2.5   40000      50      0  8.39  9.81  9.80   10.8 0.600
## 13 5     50000      50      0  9.56  10.7  10.7   12.0 0.669
## 14 5     40000      50      0  7.84  8.98  8.93   10.2 0.693

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_fit_max and filter(slices, Sigma ==
## W = 2341, p-value = 5.574e-14
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma ==
## W = 2479, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

```

```

## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 2471, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 2458, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max
## W = 2470, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max
## W = 2498, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 5.0

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```

## 
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2477, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

```

9.3 Contradictory objectives diagnostic

Here we present the results for **satisfactory trait coverage** and **activation gene coverage** found by each phenotypic fitness sharing sigma value replicate on the ordered exploitation diagnostic. Satisfactory trait coverage refers to the count of unique satisfied DIMENSIONALITY in the population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both coverage values fall between 0 and 100.

9.3.1 Satisfactory trait coverage

Satisfactory trait coverage analysis.

9.3.1.1 Coverage over time

Satisfactory trait coverage over time.

```

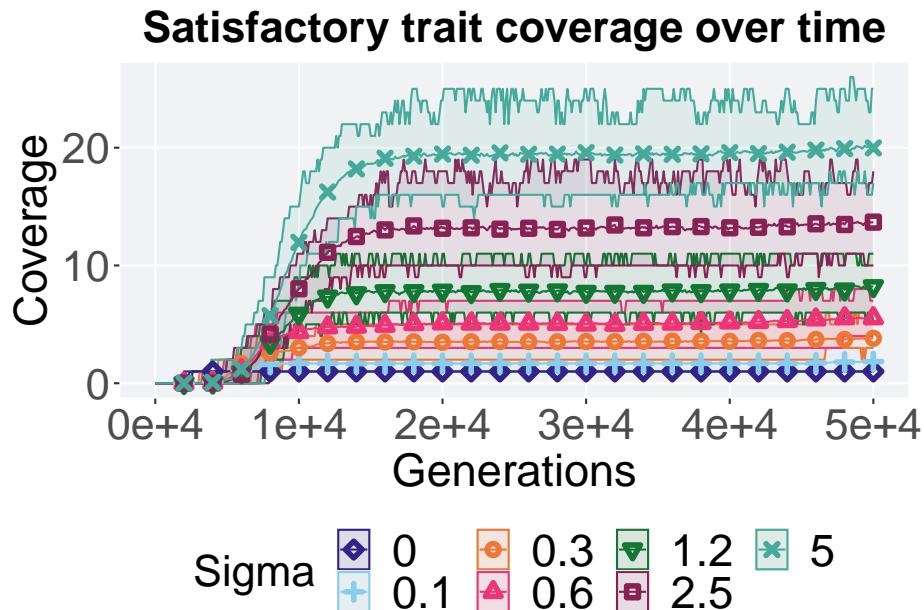
lines = filter(pfs_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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"
  ) +
  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")
  )

```

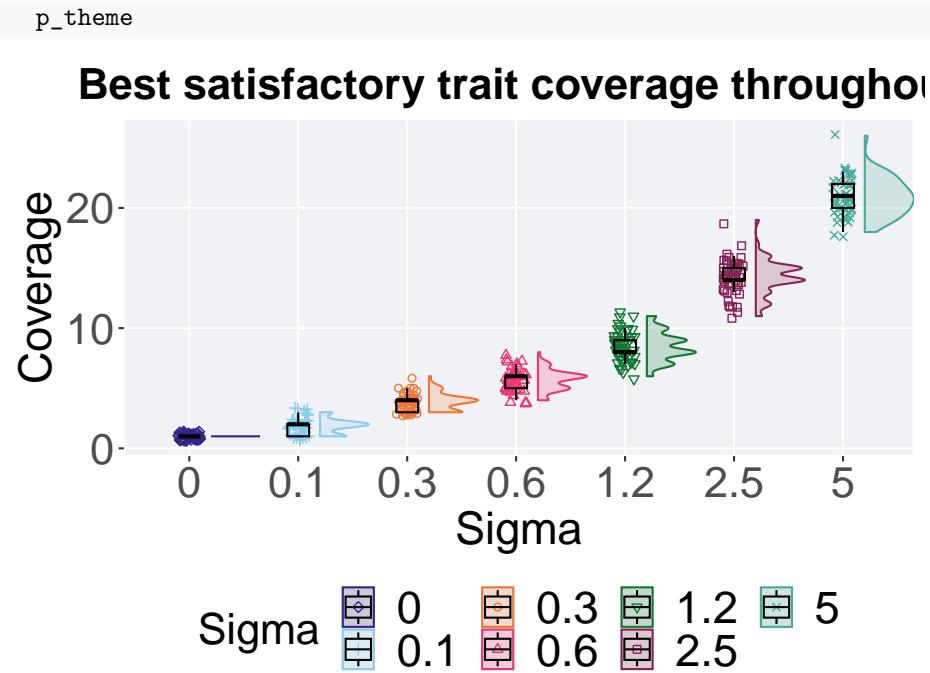
```
scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Satisfactory trait coverage over time") +
  p_theme
```



9.3.1.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
filter(pfs_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = Sigma, y = val, color = Sigma, fill = Sigma, shape = Sigma)) +
  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"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best satisfactory trait coverage throughout")+
```



9.3.1.2.1 Stats

Summary statistics for the best satisfactory trait coverage.

```
coverage = filter(pfs_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives')
  group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0      50     0     1     1     1     1     0
## 2 0.1    50     0     1     2     1.84   3     1
## 3 0.3    50     0     3     4     3.84   6     1
## 4 0.6    50     0     4     6     5.66   8     1
## 5 1.2    50     0     6     8     8.46  11     1
```

```
## 6 2.5      50      0     11     14 14.2      19      1
## 7 5      50      0     18     21 20.9      26      2
```

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

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

```
##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 338.57, df = 6, p-value < 2.2e-16
```

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

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

```
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$val and coverage$Sigma
##
##      0      0.1     0.3     0.6     1.2     2.5
## 0.1 6.8e-12 -      -      -      -      -
## 0.3 < 2e-16 3.4e-16 -      -      -      -
## 0.6 < 2e-16 < 2e-16 3.6e-13 -      -      -
## 1.2 < 2e-16 < 2e-16 < 2e-16 3.0e-15 -      -
## 2.5 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
## 5    < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
##
## P value adjustment method: bonferroni
```

9.3.1.3 End of 50,000 generations

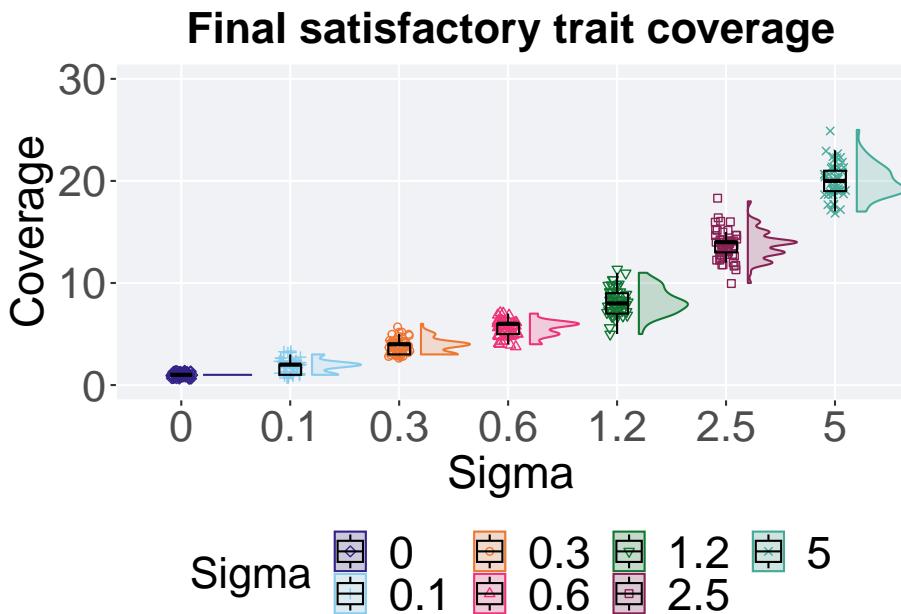
Satisfactory trait coverage in the population at the end of 50,000 generations.

```
filter(pfs_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = pop_uni_obj, color = Sigma, fill = Sigma, shape = Sigma))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5)
  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, 30)
  ) +
  scale_x_discrete()
```

```

    name="Sigma"
) +
scale_shape_manual(values=SHAPE) +
scale_colour_manual(values = cb_palette) +
scale_fill_manual(values = cb_palette) +
ggtitle("Final satisfactory trait coverage") +
p_theme

```



9.3.1.3.1 Stats

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

```

coverage = filter(pfs_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, Sigma) %>%
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: 7 x 8

```

```
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <int> <dbl>
## 1 0      50     0     1     1     1     1     0
## 2 0.1    50     0     1     2     1.84    3     1
## 3 0.3    50     0     3     4     3.82    6     1
## 4 0.6    50     0     4     6     5.54    7     1
## 5 1.2    50     0     5     8     8.22   11     2
## 6 2.5    50     0    10    14    13.7   18     1
## 7 5      50     0    17    20    20.0   25     2
```

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

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

```
##
## Kruskal-Wallis rank sum test
##
## data: pop_uni_obj by Sigma
## Kruskal-Wallis chi-squared = 338.19, df = 6, p-value < 2.2e-16
```

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

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

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$pop_uni_obj and coverage$Sigma
##
##   0     0.1    0.3    0.6    1.2    2.5
## 0.1 6.8e-12 -     -     -     -     -
## 0.3 < 2e-16 4.2e-16 -     -     -     -
## 0.6 < 2e-16 < 2e-16 8.6e-13 -     -     -
## 1.2 < 2e-16 < 2e-16 < 2e-16 5.4e-15 -     -
## 2.5 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -
## 5   < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16
##
## P value adjustment method: bonferroni
```

9.3.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the contradictory objectives diagnostic.

9.3.2.1 Coverage over time

Activation gene coverage over time.

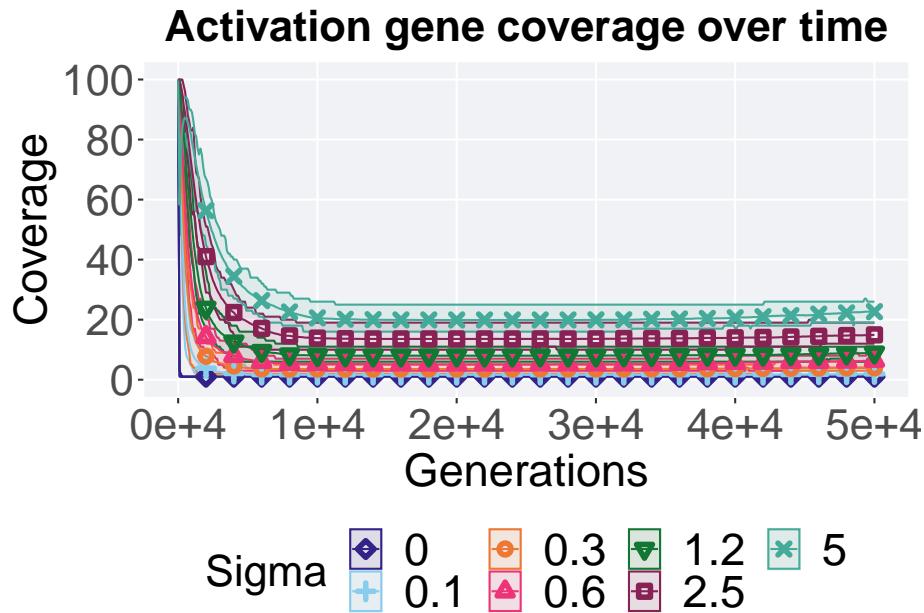
```

lines = filter(pfs_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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(-1, 101),
    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

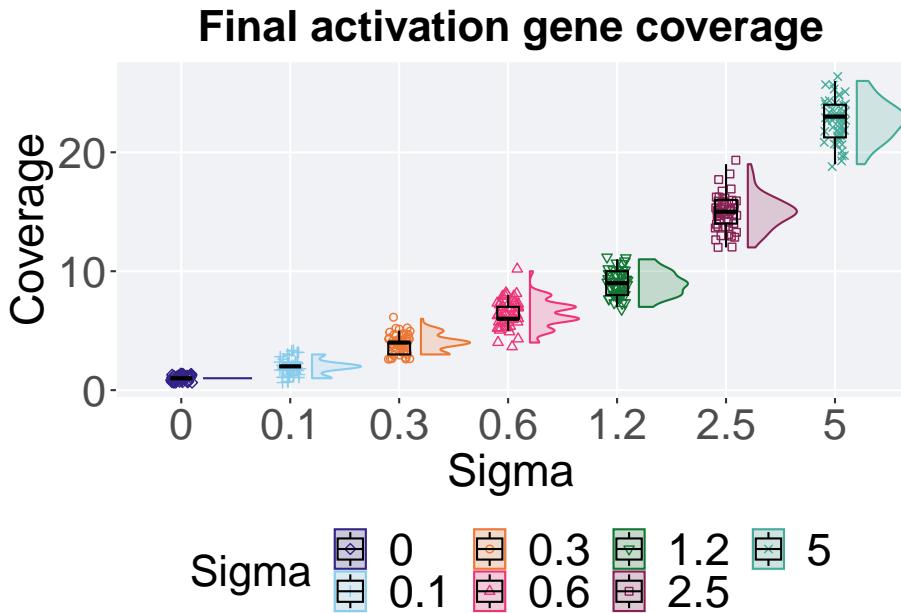
```



9.3.2.2 End of 50,000 generations

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

```
filter(pfs_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = uni_str_pos, color = Sigma, fill = Sigma, shape = Sigma,
    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"
    ) +
    scale_x_discrete(
      name="Sigma"
    ) +
    scale_shape_manual(values=SHAPE) +
    scale_colour_manual(values = cb_palette) +
    scale_fill_manual(values = cb_palette) +
    ggtitle("Final activation gene coverage") +
    p_theme
```



9.3.2.2.1 Stats

Summary statistics for the best activation gene coverage in the population at the end of 50,000 generations.

```
coverage = filter(pfs_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt  min median  mean  max  IQR
##   <dbl> <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 0      50     0     1     1     1     1     0
## 2 0.1    50     0     1     2     1.92   3     0
## 3 0.3    50     0     3     4     3.98   6     1
## 4 0.6    50     0     4     6     6.38   10    1
## 5 1.2    50     0     7     9     8.98   11    2
## 6 2.5    50     0    12    15    14.9   19    2
```

```
## 7 5      50      0     19     23 22.7     26   2.75
```

Kruskal-Wallis test provides evidence of no statistical difference among activation gene coverage in the population at the end of 50,000 generations.

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

```
##  
## Kruskal-Wallis rank sum test  
##  
## data: uni_str_pos by Sigma  
## Kruskal-Wallis chi-squared = 338.71, df = 6, p-value < 2.2e-16
```

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

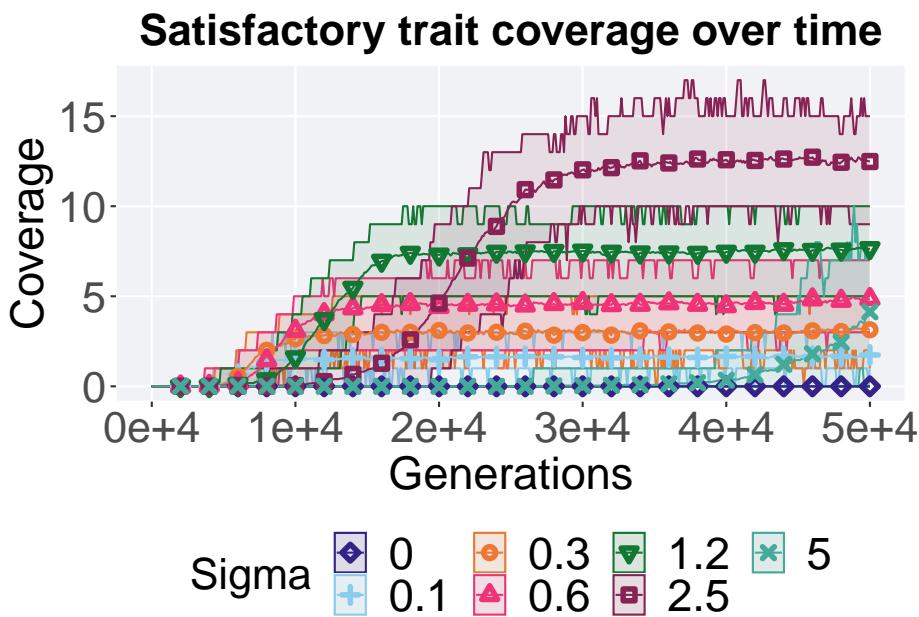
```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$Sigma, p.adjust.method =  
                      paired = FALSE, conf.int = FALSE, alternative = 'g')  
  
##  
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction  
##  
## data: coverage$uni_str_pos and coverage$Sigma  
##  
##    0     0.1     0.3     0.6     1.2     2.5  
## 0.1 2.3e-13 -      -      -      -      -  
## 0.3 < 2e-16 2.8e-16 -      -      -      -  
## 0.6 < 2e-16 < 2e-16 4.2e-14 -      -      -  
## 1.2 < 2e-16 < 2e-16 < 2e-16 1.4e-13 -      -  
## 2.5 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -  
## 5    < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 -  
##  
## P value adjustment method: bonferroni
```

9.3.3 Multi-valley crossing

9.3.3.1 Satisfactory trait coverage over time

```
lines = filter(pfs_ot_mvc, diagnostic == 'contradictory_objectives') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(pop_uni_obj),  
    mean = mean(pop_uni_obj),  
    max = max(pop_uni_obj)  
  )  
  
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## ` .groups` argument.
```

```
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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 = 1)
  scale_y_continuous(
    name="Coverage"
  ) +
  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
```



9.3.3.2 Satisfactory trait coverage comparison

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

```

# 80% and final generation comparison
end = filter(pfs_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 50000)
end$Generation <- factor(end$gen)

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

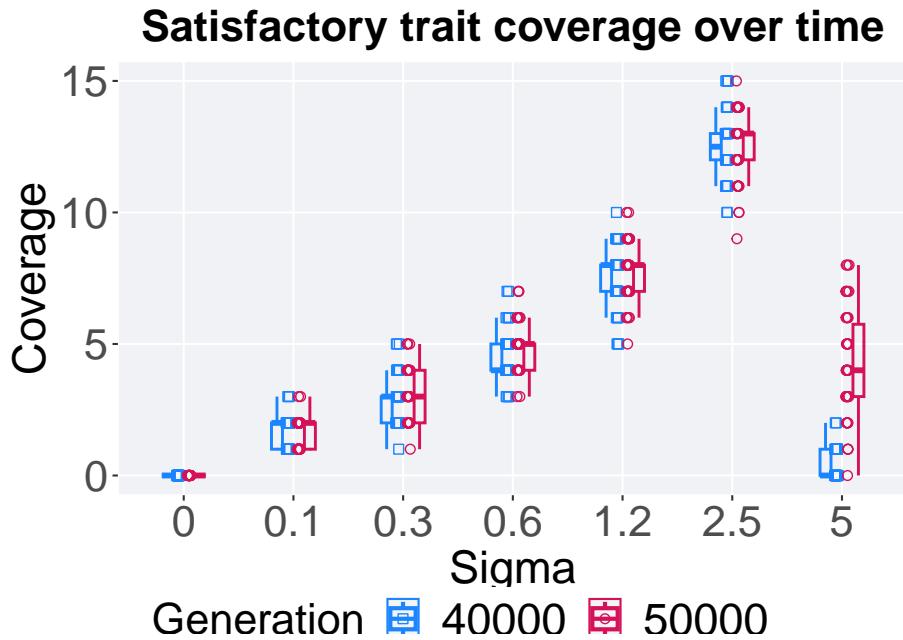
mvc_p = ggplot(mid, aes(x = Sigma, y=pop_uni_obj, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), lwd = 1, col = mvc_col[1])
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_uni_obj), col = mvc_col[2], position = position_nudge(x = 0))
  geom_boxplot(data = end, aes(x = Sigma, y=pop_uni_obj), position = position_nudge(x = 0))

  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
  )+
  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 over time") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



9.3.3.2.1 Stats

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

```
slices = filter(pfs_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.
## # A tibble: 14 x 9
## # Groups:   Sigma [7]
##   Sigma Generation count na_cnt  min median  mean  max    IQR
##   <fct> <fct>     <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
```

```

##   1 0    50000      50      0      0      0      0      0  0
##   2 0    40000      50      0      0      0      0      0  0
##   3 0.1  50000      50      0      1      2     1.74     3  1
##   4 0.1  40000      50      0      1      2     1.66     3  1
##   5 0.3   50000      50      0      1      3     3.14     5  2
##   6 0.3   40000      50      0      1      3     2.9      5  1
##   7 0.6   50000      50      0      3      5     4.86     7  1
##   8 0.6   40000      50      0      3      4     4.46     7  1
##   9 1.2   50000      50      0      5      8     7.66    10  1
##  10 1.2  40000      50      0      5      8     7.46    10  1
##  11 2.5   50000      50      0      9     13    12.5    15  1
##  12 2.5   40000      50      0     10    12.5   12.6    15  1
##  13 5    50000      50      0      0      4     4.14     8 2.75
##  14 5    40000      50      0      0      0     0.34     2  1

Sigma 0.0

wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1

wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.1 & Generation == 40000)$pop_uni_obj
## W = 1352.5, p-value = 0.414
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

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

```

```
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.3 & Generation == 40000)$pop_uni_obj
## W = 1423.5, p-value = 0.2118
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_uni_obj
## W = 1572, p-value = 0.01868
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_uni_obj
## W = 1339, p-value = 0.5259
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_uni_obj
## W = 1242.5, p-value = 0.9603
## alternative hypothesis: true location shift is not equal to 0

Sigma 5.0

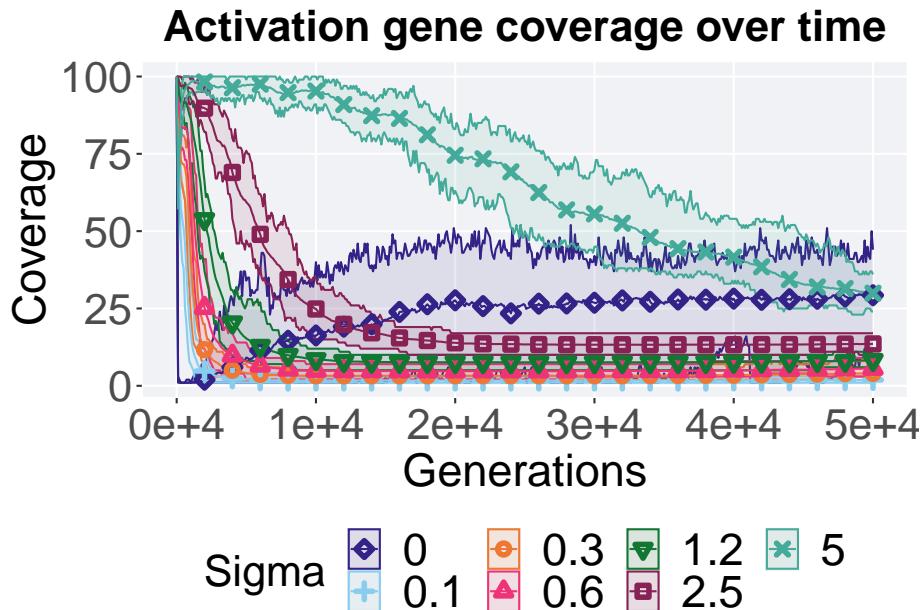
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

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

```
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_uni_obj and filter(slices,  
## W = 2436, p-value < 2.2e-16  
## alternative hypothesis: true location shift is not equal to 0
```

```
lines = filter(pfs_ot_mvc, diagnostic == 'contradictory_objectives') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(uni_str_pos),  
    mean = mean(uni_str_pos),  
    max = max(uni_str_pos)  
  )
```

```
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## ` `.groups` argument.  
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +  
  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"  
  ) +  
  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
```



9.3.3.4 Activation gene coverage comparison

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

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

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

mvc_p = ggplot(mid, aes(x = Sigma, y=uni_str_pos, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 100) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])
  #geom_point(data = end, aes(x = Sigma, y=uni_str_pos), col = mvc_col[2], position = position_jitternudge(jitter.width = .03, nudge.x = 0.15), size = 100) +
  #geom_boxplot(data = end, aes(x = Sigma, y=uni_str_pos), position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

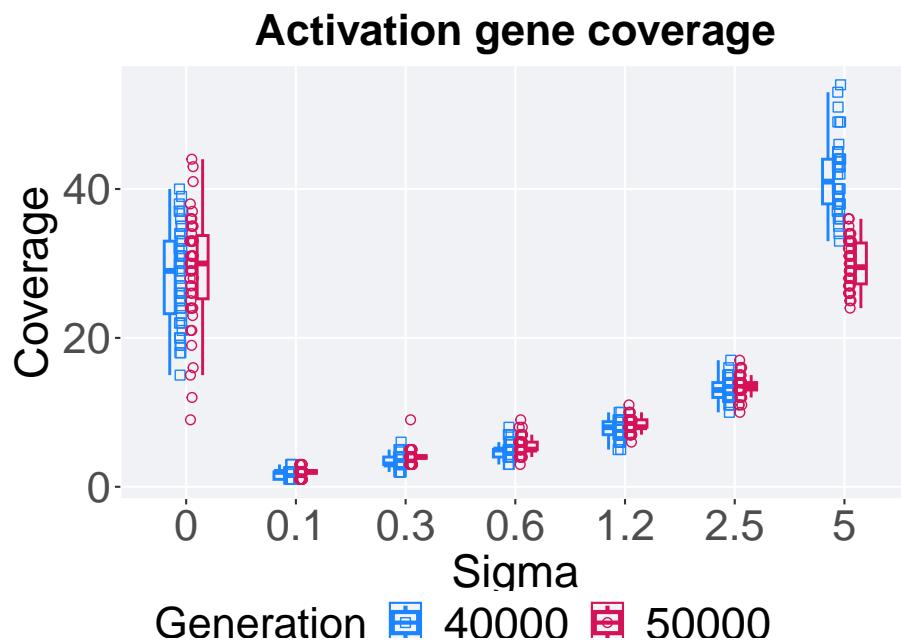
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
  )+
  scale_shape_manual(values=c(0,1))+
```

```

scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
p_theme

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

```



9.3.3.4.1 Stats

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

```

slices = filter(pfs_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 |
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <dbl> <dbl> <int> <dbl>
## 1 0     50000      50     0     9    30  29.3    44   8.5
## 2 0     40000      50     0    15    29  28.2    40  9.75
## 3 0.1   50000      50     0     1     2   1.88    3    0
## 4 0.1   40000      50     0     1     2   1.74    3    1
## 5 0.3   50000      50     0     3     4   4.02    9    0
## 6 0.3   40000      50     0     2     3   3.42    6    1
## 7 0.6   50000      50     0     3     5   5.58    9    1
## 8 0.6   40000      50     0     3     5     5     8    1
## 9 1.2   50000      50     0     6     8   8.28   11    1
## 10 1.2  40000      50     0     5     8   7.72   10  1.75
## 11 2.5   50000      50     0    10  13.5  13.5    17    1
## 12 2.5  40000      50     0    10  13.3  13.3    17    2
## 13 5    50000      50     0    24  29.5  29.9    36   5.5
## 14 5    40000      50     0   33   41   41.6    54    6

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =
## W = 1391, p-value = 0.332
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##

```

```

## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1395, p-value = 0.2428
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1709, p-value = 0.0006733
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1616, p-value = 0.007901
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1575.5, p-value = 0.01947
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$uni_str_pos,
            alternative = 't')

```

```

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  

## W = 1391, p-value = 0.3181  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 5.0  

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, Sigma == 5.0 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  

## W = 24.5, p-value < 2.2e-16  

## alternative hypothesis: true location shift is not equal to 0

```

9.4 Multi-path exploration results

Here we present the results for **best performances** and **activation gene coverage** found by each phenotypic fitness sharing sigma value replicate on the multi-path exploration diagnostic. Best performance found refers to the largest average trait score found in a given population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both values fall between 0 and 100.

9.4.1 Performance

Here we analyze the performances for each parameter replicate on the multi-path exploration diagnostic.

9.4.1.1 Performance over time

Performance over time.

```

lines = filter(pfs_ot, diagnostic == 'multipath_exploration') %>%  

  group_by(Sigma, gen) %>%  

  dplyr::summarise(  

    min = min(pop_fit_max),  

    mean = mean(pop_fit_max),  

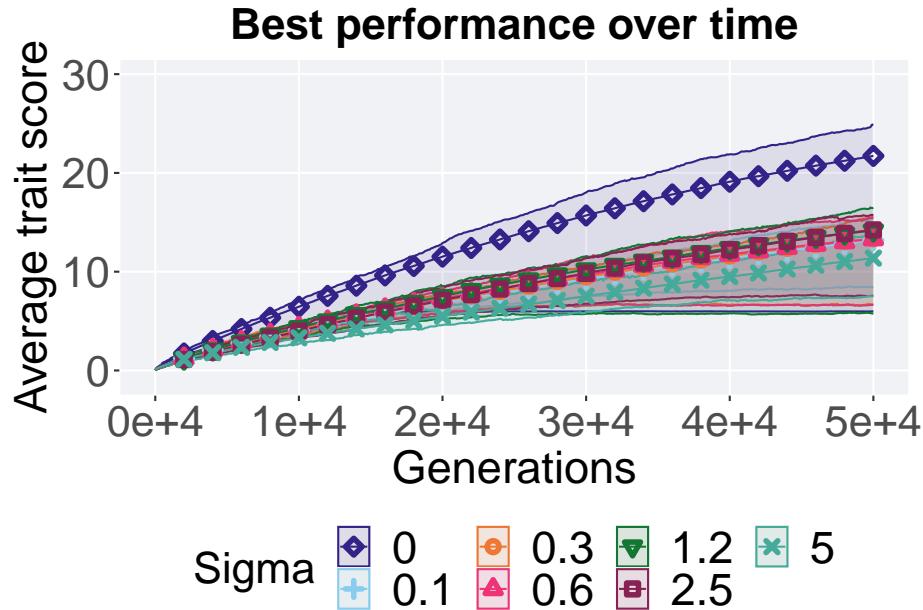
    max = max(pop_fit_max)
  )  

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

## `.` argument.

```

```
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma))
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2)
  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(-1, 30)
  ) +
  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("Best performance over time") +
  p_theme
```

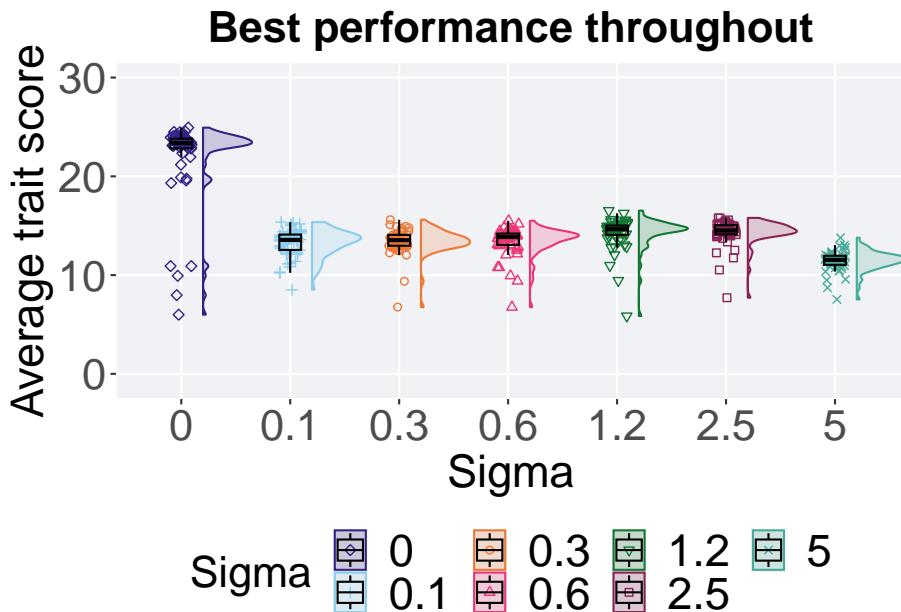


9.4.1.2 Best performance throughout

Here we plot the performance of the best performing solution found throughout 50,000 generations.

```

filter(pfs_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
    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, 30)
    ) +
    scale_x_discrete(
      name = "Sigma"
    ) +
    scale_shape_manual(values = SHAPE) +
    scale_colour_manual(values = cb_palette) +
    scale_fill_manual(values = cb_palette) +
    ggtitle("Best performance throughout") +
    p_theme
  
```



9.4.1.2.1 Stats

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

```

performance = filter(pfs_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration')
group_by(performance, Sigma) %>%
  
```

```
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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0     50     0    5.99  23.4  21.8  24.9  0.986
## 2 0.1   50     0    8.51  13.6  13.2  15.4  1.55
## 3 0.3   50     0    6.76  13.5  13.4  15.6  1.10
## 4 0.6   50     0    6.75  13.9  13.4  15.5  1.14
## 5 1.2   50     0    5.86  14.6  14.2  16.5  0.949
## 6 2.5   50     0    7.71  14.5  14.3  15.8  0.925
## 7 5     50     0    7.54  11.5  11.4  13.8  0.914
```

Kruskal–Wallis test provides evidence of statistical differences among the best performing solution found throughout 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 175.44, df = 6, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performing solution found throughout 50,000 generations.

```
pairwise.wilcox.test(x = performance$val, g = performance$Sigma, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')

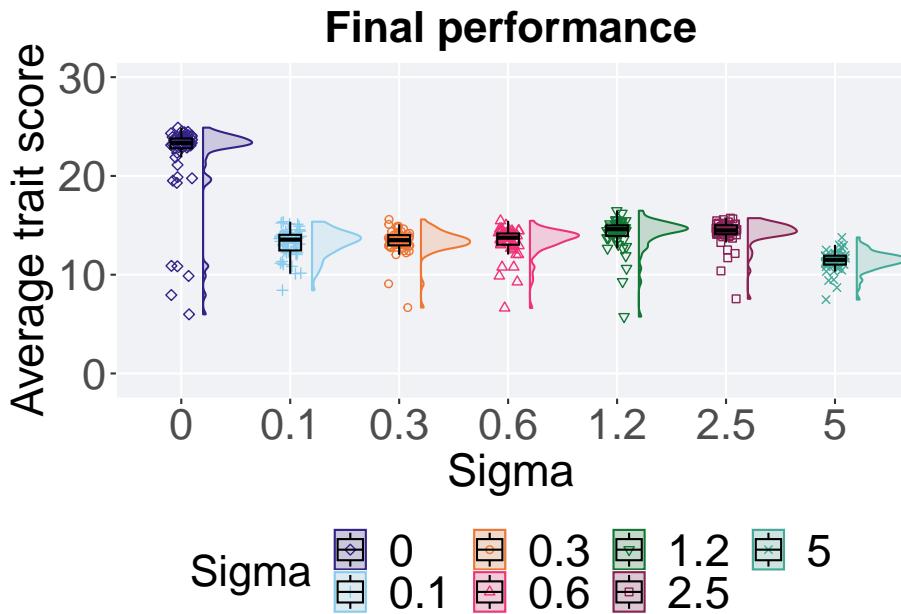
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
##          0      0.1     0.3     0.6     1.2     2.5
## 0.1 7.5e-11 -       -       -       -       -
## 0.3 8.3e-11 1.00000 -       -       -       -
## 0.6 6.2e-11 1.00000 1.00000 -       -       -
## 1.2 7.9e-11 0.00014 0.00020 0.00019 -       -
```

```
## 2.5 8.7e-11 3.9e-05 2.8e-05 7.0e-05 1.00000 -
## 5 3.1e-11 1.1e-08 2.4e-12 3.9e-10 1.3e-12 1.9e-13
##
## P value adjustment method: bonferroni
```

9.4.1.3 End of 50,000 generations

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

```
filter(pfs_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = pop_fit_max / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma))
  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, 30)
  ) +
  scale_x_discrete(
    name = "Sigma"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final performance") +
  p_theme
```



9.4.1.3.1 Stats

Summary statistics for the best performance in the population at the end of 50,000 generations.

```
performance = filter(pfs_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(performance, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0         50      0  5.99  23.3  21.7  24.9  0.997
## 2 0.1       50      0  8.42  13.6  13.2  15.4  1.58 
## 3 0.3       50      0  6.67  13.5  13.3  15.6  1.04 
## 4 0.6       50      0  6.63  13.7  13.3  15.5  1.14 
## 5 1.2       50      0  5.76  14.6  14.2  16.5  1.08 
## 6 2.5       50      0  7.56  14.5  14.2  15.7  0.915
## 7 5         50      0  7.48  11.5  11.4  13.8  0.898
```

Kruskal–Wallis test provides evidence of statistical differences among best performance in the population at the end of 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: pop_fit_max by Sigma
## Kruskal-Wallis chi-squared = 175.09, df = 6, p-value < 2.2e-16
```

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

```
pairwise.wilcox.test(x = performance$pop_fit_max, g = performance$Sigma , p.adjust.method = 'bonferroni',
                      paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$pop_fit_max and performance$Sigma
```

```
##
##      0      0.1      0.3      0.6      1.2      2.5
## 0.1 7.5e-11 -      -      -      -
## 0.3 8.3e-11 1e+00 -      -      -      -
## 0.6 6.2e-11 1e+00 1e+00 -      -      -
## 1.2 7.2e-11 2e-04 2e-04 2e-04 -      -
## 2.5 8.7e-11 3.4e-05 2.1e-05 3.9e-05 1e+00 -
## 5   3.1e-11 1.3e-08 2.4e-12 4.9e-10 2.2e-12 2.1e-13
##
## P value adjustment method: bonferroni
```

9.4.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the multi-path exploration diagnostic.

9.4.2.1 Coverage over time

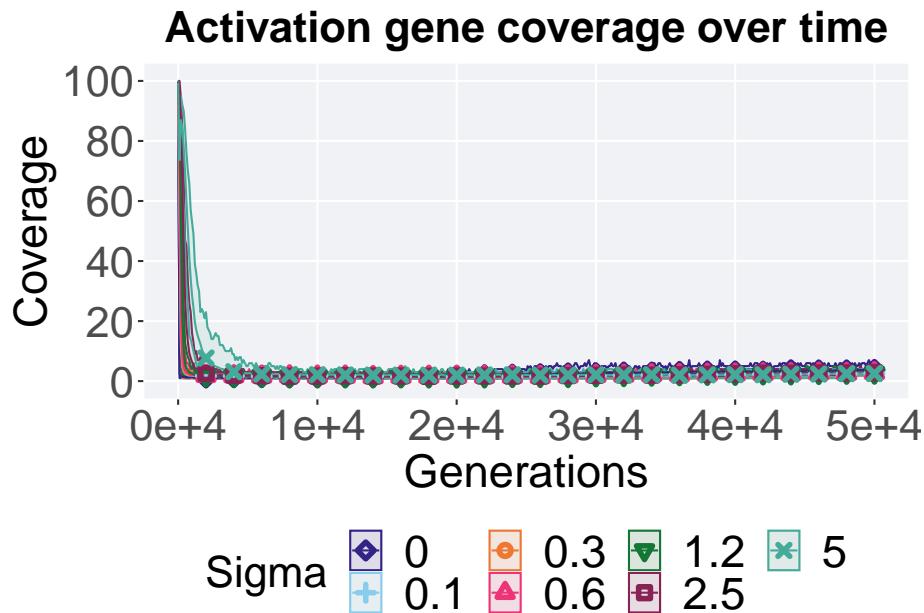
Activation gene coverage over time.

```
lines = filter(pfs_ot, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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(-1, 101),
    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")
```

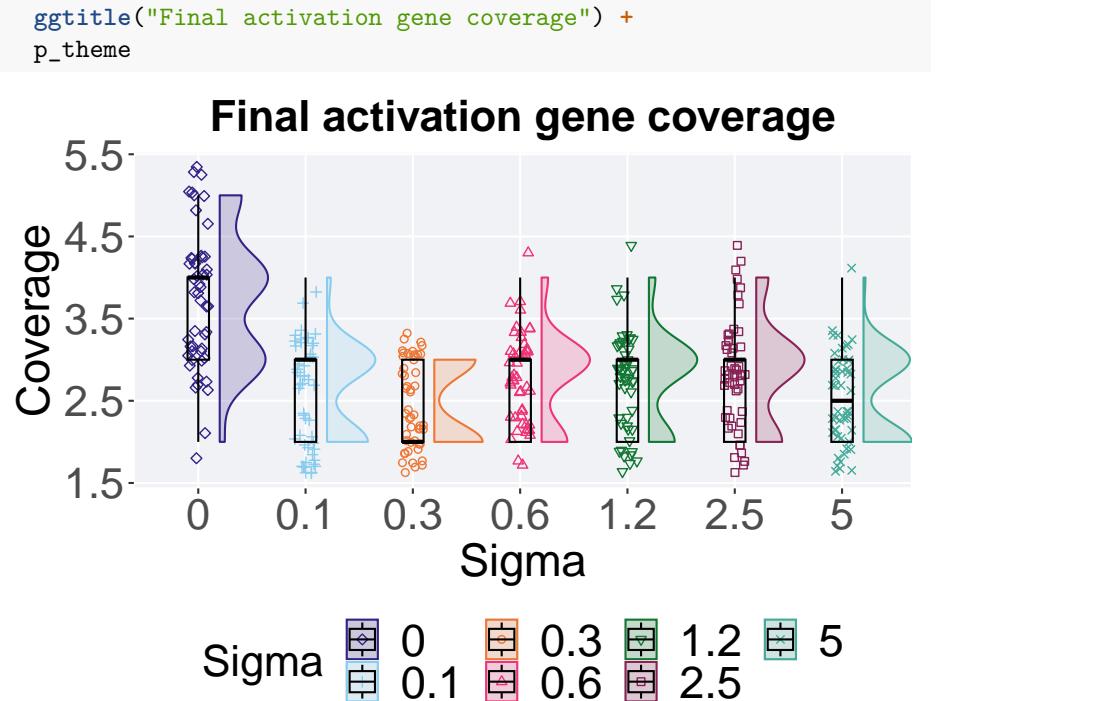
```
) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Activation gene coverage over time") +
  p_theme
```



9.4.2.2 End of 50,000 generations

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

```
filter(pfs_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = uni_str_pos, color = Sigma, fill = Sigma, shape = Sigma)) +
  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"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
```



9.4.2.2.1 Stats

Summary statistics for the activation gene coverage in the population at the end of 50,000 generations.

```
coverage = filter(pfs_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt  min median  mean  max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 0        50      0     2     4    3.72     5     1
## 2 0.1      50      0     2     3    2.6      4     1
## 3 0.3      50      0     2     2    2.46     3     1
```

```
## 4 0.6      50     0     2     3    2.76     4     1
## 5 1.2      50     0     2     3    2.76     4     1
## 6 2.5      50     0     2     3    2.84     4     1
## 7 5        50     0     2    2.5   2.52     4     1
```

Kruskal–Wallis test provides evidence of statistical differences among activation gene coverage in the population at the end of 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: uni_str_pos by Sigma
## Kruskal-Wallis chi-squared = 81.793, df = 6, p-value = 1.522e-15
```

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

```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$Sigma , p.adjust.method =
  paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$uni_str_pos and coverage$Sigma
##
##          0       0.1      0.3      0.6      1.2      2.5
## 0.1 4.4e-09 -       -       -       -       -
## 0.3 5.8e-11 1.000 -       -       -       -
## 0.6 2.9e-07 1.000 0.250 -       -       -
## 1.2 2.9e-07 1.000 0.250 1.000 -       -
## 2.5 5.8e-06 1.000 0.069 1.000 1.000 -
## 5   4.3e-10 1.000 1.000 0.915 0.915 0.283
##
## P value adjustment method: bonferroni
```

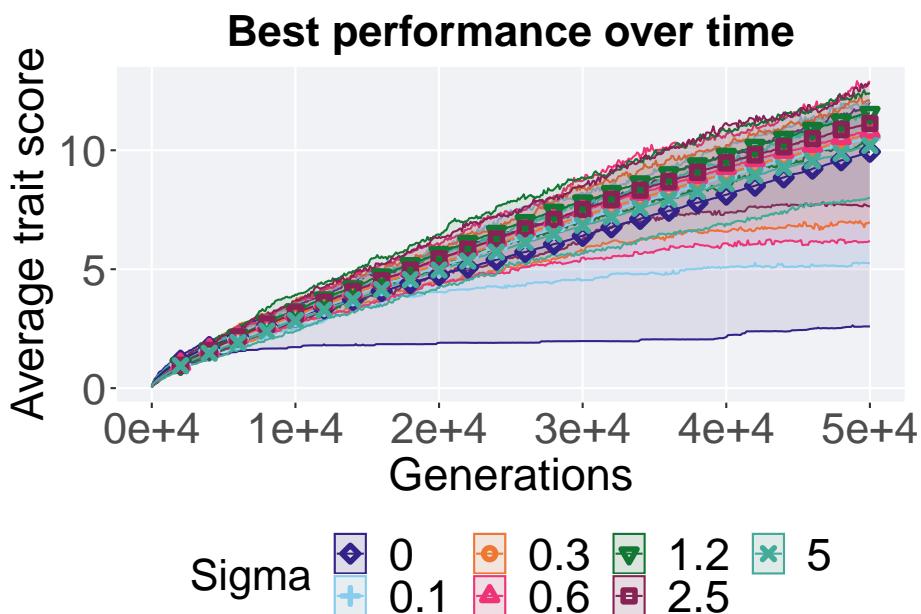
9.4.3 Multi-valley crossing

9.4.3.1 Performance over time

```
lines = filter(pfs_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )
```

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

```
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +  
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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 = 1)  
  scale_y_continuous(  
    name="Average trait score"  
) +  
  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("Best performance over time") +  
  p_theme
```



9.4.3.2 Performance comparisons

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

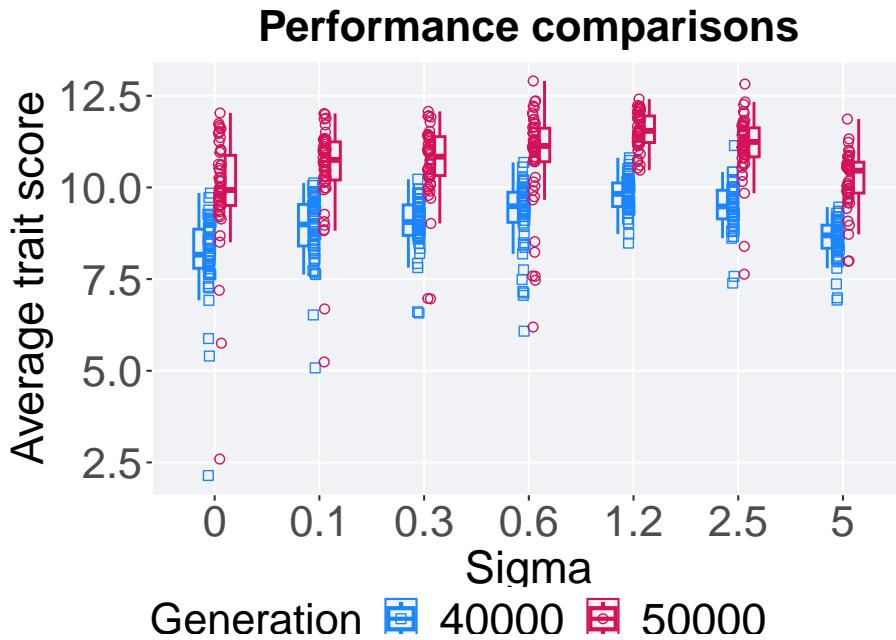
mid = filter(pfs_ot_mvc, diagnostic == 'multipath_exploration' & gen == 40000)
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = mvc_col[1]))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), lwd = 1.5, size = 100)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2])
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 0), col = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  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
)
```



9.4.3.2.1 Stats

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

```
# 80% and final generation comparison
slices = filter(pfs_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.`groups` argument.

## # A tibble: 14 x 9
## # Groups:   Sigma [7]
##   Sigma Generation count  na_cnt   min  median   mean   max    IQR
##   <dbl>     <dbl>   <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

##   <fct> <fct>     <int>  <int> <dbl>  <dbl> <dbl> <dbl>
## 1 0    50000      50     0  2.59   9.93  9.94 12.0  1.37
## 2 0    40000      50     0  2.14   8.17  8.13  9.85 1.07
## 3 0.1  50000      50     0  5.24  10.8  10.5  12.0  1.04
## 4 0.1  40000      50     0  5.08   8.99  8.87 10.1  1.13
## 5 0.3  50000      50     0  6.96  10.8  10.7  12.1  1.06
## 6 0.3  40000      50     0  6.57   9.06  9.01 10.2  0.832
## 7 0.6  50000      50     0  6.19  11.1  10.8  12.9  0.913
## 8 0.6  40000      50     0  6.08   9.49  9.26 10.7  0.821
## 9 1.2  50000      50     0 10.5   11.5  11.6  12.4  0.724
## 10 1.2 40000      50     0  8.48   9.82  9.80 10.8  0.643
## 11 2.5  50000      50     0  7.63  11.2  11.1  12.8  0.791
## 12 2.5  40000      50     0  7.39   9.48  9.47 11.1  0.781
## 13 5    50000      50     0  7.99  10.5  10.2  11.9  0.842
## 14 5    40000      50     0  6.93   8.69  8.59  9.46  0.629

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0 & Generation == 40000)$pop_fit_max
## W = 2280, p-value = 1.273e-12
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max
## W = 2268, p-value = 2.308e-12
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```
##  
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  
## W = 2358, p-value = 2.26e-14  
## alternative hypothesis: true location shift is not equal to 0  
  
Sigma 0.6  
  
wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,  
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,  
            alternative = 't')  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  
## W = 2168, p-value = 2.531e-10  
## alternative hypothesis: true location shift is not equal to 0  
  
Sigma 1.2  
  
wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,  
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,  
            alternative = 't')  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  
## W = 2487, p-value < 2.2e-16  
## alternative hypothesis: true location shift is not equal to 0  
  
Sigma 2.5  
  
wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,  
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,  
            alternative = 't')  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  
## W = 2360, p-value = 2.03e-14  
## alternative hypothesis: true location shift is not equal to 0  
  
Sigma 5.0  
  
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,  
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,  
            alternative = 't')  
  
##
```

```

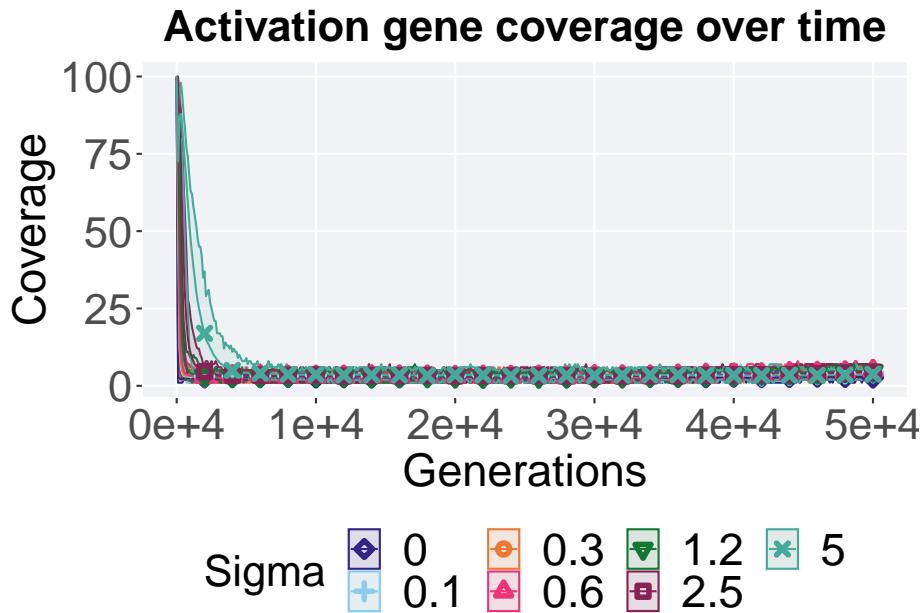
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 2364, p-value = 1.638e-14
## alternative hypothesis: true location shift is not equal to 0

lines = filter(pfs_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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"
  ) +
  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

```



9.4.3.4 Activation gene coverage comparison

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

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

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

mvc_p = ggplot(mid, aes(x = Sigma, y=uni_str_pos, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 1) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=uni_str_pos), col = mvc_col[2], position = position_jitternudge(jitter.width = .03, nudge.x = 0.15), size = 1) +
  geom_boxplot(data = end, aes(x = Sigma, y=uni_str_pos), position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

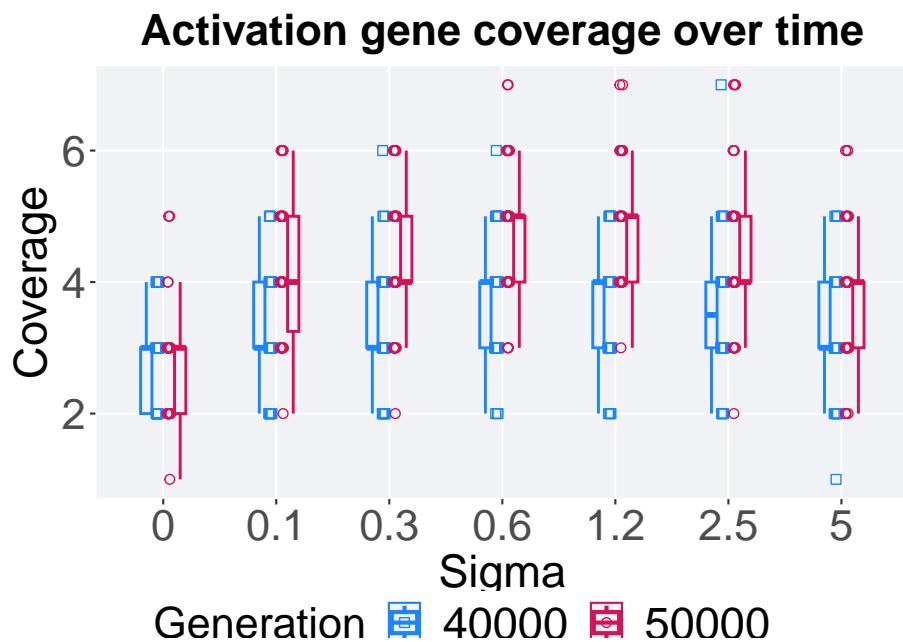
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
) +
  scale_shape_manual(values=c(0,1)) +
```

```

scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
p_theme

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

```



9.4.3.4.1 Stats

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

```

slices = filter(pfs_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen ==
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <dbl> <dbl> <int> <dbl>
## 1 0      50000      50    0     1     3    2.64    5    1
## 2 0      40000      50    0     2     3    2.9     4    1
## 3 0.1    50000      50    0     2     4    4.28    6   1.75
## 4 0.1    40000      50    0     2     3    3.38    5    1
## 5 0.3    50000      50    0     2     4    4.34    6    1
## 6 0.3    40000      50    0     2     3    3.28    6    1
## 7 0.6    50000      50    0     3     5    4.9     7    1
## 8 0.6    40000      50    0     2     4    3.72    6    1
## 9 1.2    50000      50    0     3     5    4.8     7    1
## 10 1.2   40000      50    0     2     4    3.62    5    1
## 11 2.5    50000      50    0     2     4    4.38    7    1
## 12 2.5   40000      50    0     2     3.5   3.56    7    1
## 13 5     50000      50    0     2     4    3.74    6    1
## 14 5     40000      50    0     1     3    3.36    5    1

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =
## W = 997.5, p-value = 0.05888
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##

```

```

## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1823, p-value = 4.209e-05
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1914.5, p-value = 2.023e-06
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 2024, p-value = 3.029e-08
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 2027.5, p-value = 1.962e-08
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$uni_str_pos,
            alternative = 't')

```

```
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  
## W = 1793.5, p-value = 8.472e-05  
## alternative hypothesis: true location shift is not equal to 0  
Sigma 5.0  
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$uni_str_pos,  
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$uni_str_pos,  
            alternative = 't')  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  
## W = 1468, p-value = 0.1152  
## alternative hypothesis: true location shift is not equal to 0
```


Chapter 10

Nondominated sorting

We present the results from our parameter sweep on nondominated sorting. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, performance with no similarity penalty and stochastic remainder selection is used to identify parent solutions.

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

10.1 Exploitation rate results

Here we present the results for **best performances** found by each nondominated sorting sigma value replicate on the exploitation rate diagnostic. Best performance found refers to the largest average trait score found in a given population. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, no similarity penalty is used, and only stochastic remainder selection is used to identify parent solutions.

10.1.1 Performance over time

Performance over time.

```
lines = filter(nds_ot, diagnostic == 'exploitation_rate') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
```

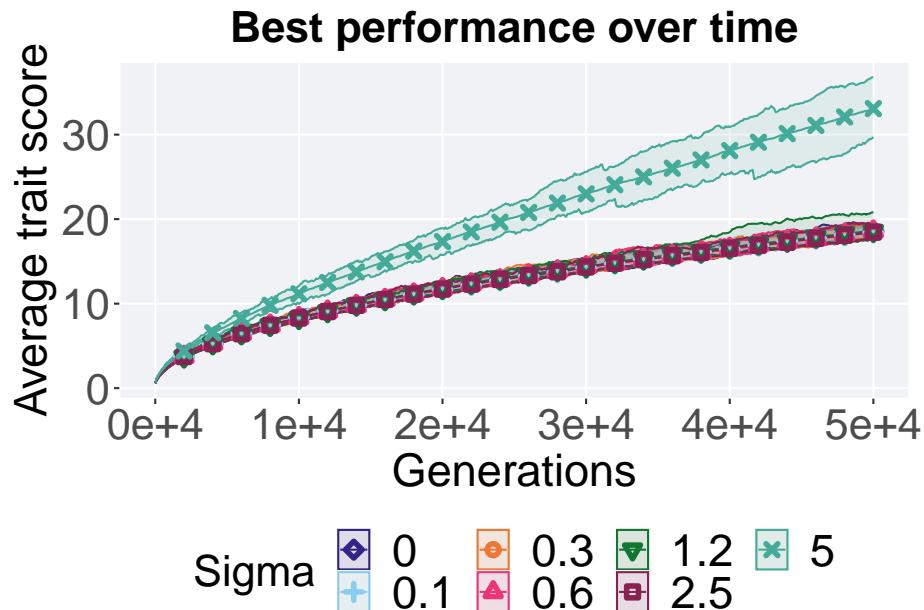
```

    max = max(pop_fit_max)
)

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

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color =
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2) +
  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"
  ) +
  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("Best performance over time") +
  p_theme

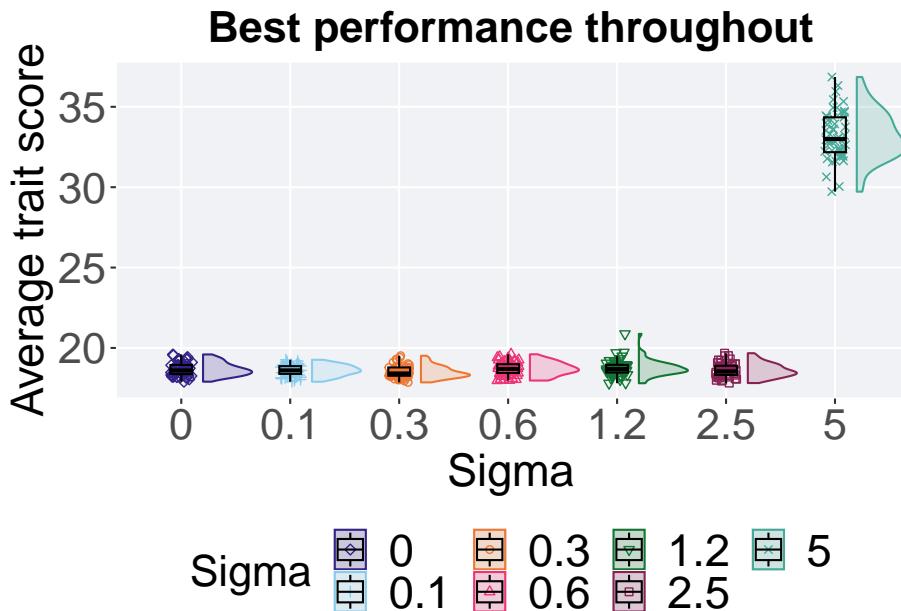
```



10.1.2 Best performance throughout

The best performance found throughout 50,000 generations.

```
filter(nds_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
  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"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best performance throughout") +
  p_theme
```



10.1.2.1 Stats

Summary statistics for the best performance found.

```

performance = filter(nds_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate'
group_by(performance, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0      50     0  17.9  18.6  18.7  19.6  0.568
## 2 0.1    50     0  17.9  18.6  18.6  19.3  0.482
## 3 0.3    50     0  17.9  18.4  18.5  19.5  0.516
## 4 0.6    50     0  18.0  18.7  18.7  19.6  0.542
## 5 1.2    50     0  17.8  18.7  18.8  20.9  0.481
## 6 2.5    50     0  17.8  18.5  18.6  19.7  0.572
## 7 5      50     0  29.7  33.0  33.2  36.9  2.17

```

Kruskal–Wallis test provides evidence of significant differences among sigma values on the best performance found.

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

```

##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 135.65, df = 6, p-value < 2.2e-16

```

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

```

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

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
##    0    0.1   0.3   0.6   1.2   2.5
## 0.1 1.00 - - - - -
## 0.3 0.47 1.00 - - - - -

```

```

## 0.6 1.00 1.00 1.00 - - -
## 1.2 1.00 1.00 1.00 1.00 - - -
## 2.5 1.00 1.00 1.00 1.00 1.00 -
## 5 1.00 1.00 1.00 1.00 1.00 1.00
##
## P value adjustment method: bonferroni

```

10.1.3 Multi-valley crossing

10.1.3.1 Performance comparison

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

```

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

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

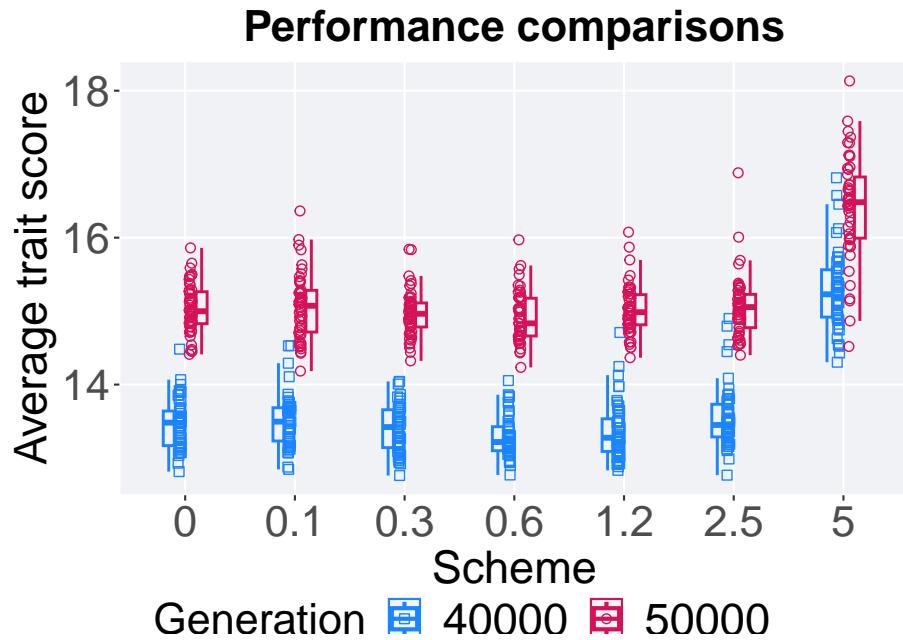
mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = Generation))
  geom_point(col = mvc_col[1], position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 10)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = 0.15, y = 0), size = 10)
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = 0.15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  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
)

```



10.1.3.2 Stats

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

```
slices = filter(nds_ot_mvc, diagnostic == 'exploitation_rate' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.
## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
```

```

## 1 0      50000      50      0 14.4   15.0  15.0  15.9 0.435
## 2 0      40000      50      0 12.8   13.5  13.5  14.5 0.471
## 3 0.1    50000      50      0 14.2   15.1  15.1  16.4 0.568
## 4 0.1    40000      50      0 12.8   13.5  13.5  14.5 0.452
## 5 0.3    50000      50      0 14.3   15.0  15.0  15.8 0.327
## 6 0.3    40000      50      0 12.8   13.4  13.4  14.0 0.516
## 7 0.6    50000      50      0 14.2   14.8  14.9  16.0 0.514
## 8 0.6    40000      50      0 12.8   13.2  13.3  14.1 0.327
## 9 1.2    50000      50      0 14.4   15.0  15.0  16.1 0.411
## 10 1.2   40000      50      0 12.8   13.3  13.4  14.7 0.444
## 11 2.5    50000      50      0 14.4   15.1  15.1  16.9 0.454
## 12 2.5   40000      50      0 12.8   13.5  13.5  14.9 0.444
## 13 5     50000      50      0 14.5   16.5  16.4  18.1 0.832
## 14 5     40000      50      0 14.3   15.2  15.3  16.8 0.644

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2496, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2485, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

```

```

## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6
wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 2500, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2
wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max
## W = 2493, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5
wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max
## W = 2465, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 5.0
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =  
## W = 2232, p-value = 1.321e-11  
## alternative hypothesis: true location shift is not equal to 0
```

10.2 Ordered exploitation results

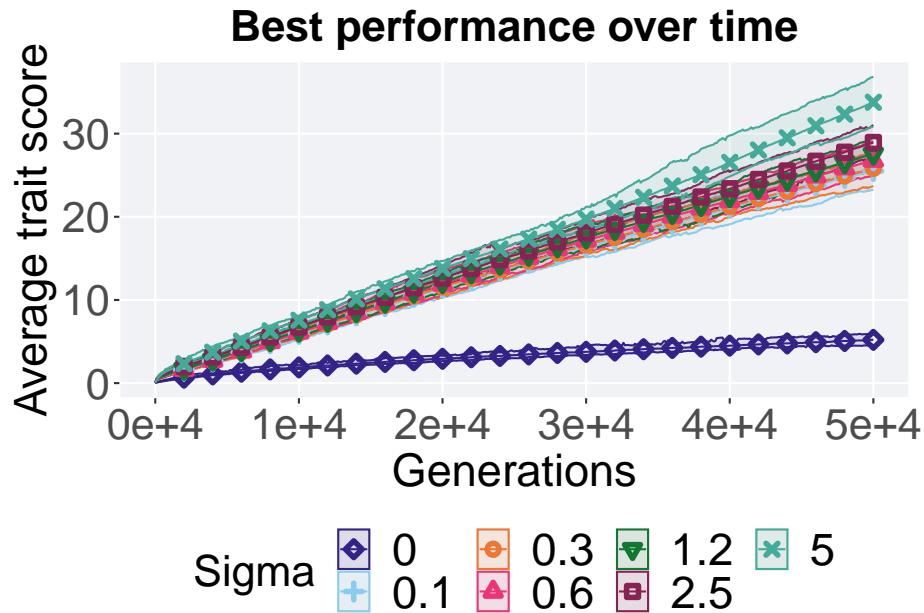
Here we present the results for **best performances** found by each nondominated sorting sigma value replicate on the ordered exploitation diagnostic. Best performance found refers to the largest average trait score found in a given population. 50 replicates are conducted for each sigma parameter value explored. Note that when `sigma = 0.0`, no similarity penalty is used, and only stochastic remainder selection is used to identify parent solutions.

10.2.1 Performance over time

Performance over time.

```
lines = filter(nds_ot, diagnostic == 'ordered_exploitation') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(pop_fit_max),  
    mean = mean(pop_fit_max),  
    max = max(pop_fit_max)  
  )  
  
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## `.` argument.  
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma, sha  
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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="Average trait score"  
  ) +  
  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) +
```

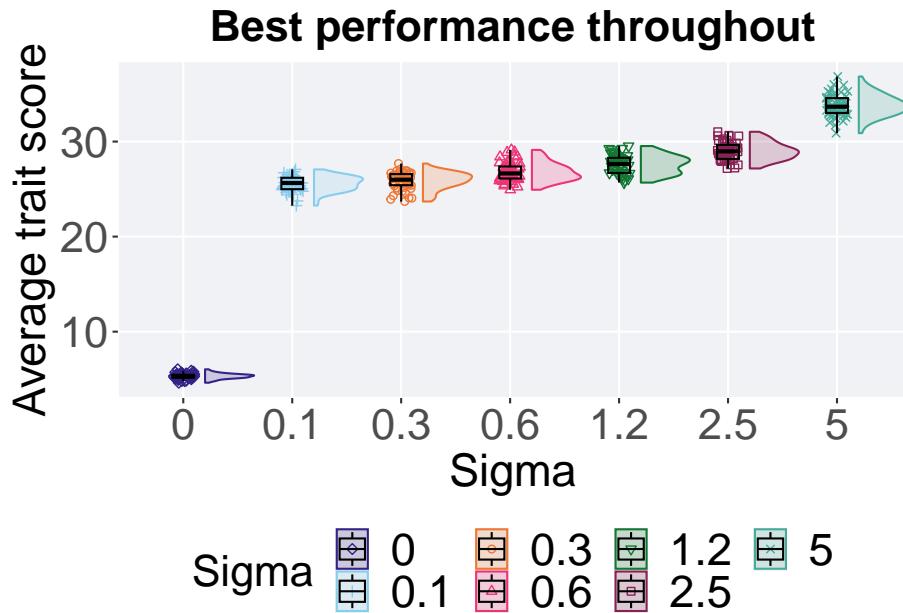
```
ggtile("Best performance over time") +
  p_theme
```



10.2.2 Best performance throughout

The best performance found throughout 50,000 generations.

```
filter(nds_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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"
  ) +
  scale_x_discrete(
    name = "Sigma"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtile("Best performance throughout") +
  p_theme
```



10.2.2.1 Stats

Summary statistics about the best performance found.

```
performance = filter(nds_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation')
group_by(performance, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0      50     0  4.63  5.32  5.32  6.04  0.313
## 2 0.1    50     0 23.3   25.6  25.6  27.1  1.20 
## 3 0.3    50     0 23.7   26.0  25.9  27.7  1.17 
## 4 0.6    50     0 24.9   26.6  26.8  29.1  1.26 
## 5 1.2    50     0 25.7   27.6  27.6  29.5  1.59 
## 6 2.5    50     0 27.2   29.0  29.0  31.0  1.47 
## 7 5      50     0 30.9   33.7  33.8  36.8  1.56
```

Kruskal–Wallis test provides evidence of statistical differences for the best performance found in the population throughout 50,000 generations.

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

##
##  Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 301.08, df = 6, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performance found in the population throughout 50,000 generations.

```
pairwise.wilcox.test(x = performance$val, g = performance$Sigma , p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = '1')

##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
##      0 0.1 0.3 0.6 1.2 2.5
## 0.1 1 - - - - -
## 0.3 1 1 - - - - -
## 0.6 1 1 1 - - - -
## 1.2 1 1 1 1 - - -
## 2.5 1 1 1 1 1 - -
## 5   1 1 1 1 1 1
```


P value adjustment method: bonferroni

10.2.3 Multi-valley crossing

10.2.3.1 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(nds_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 50000)
end$Generation <- factor(end$gen)

mid = filter(nds_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 40000)
mid$Generation <- factor(mid$gen)

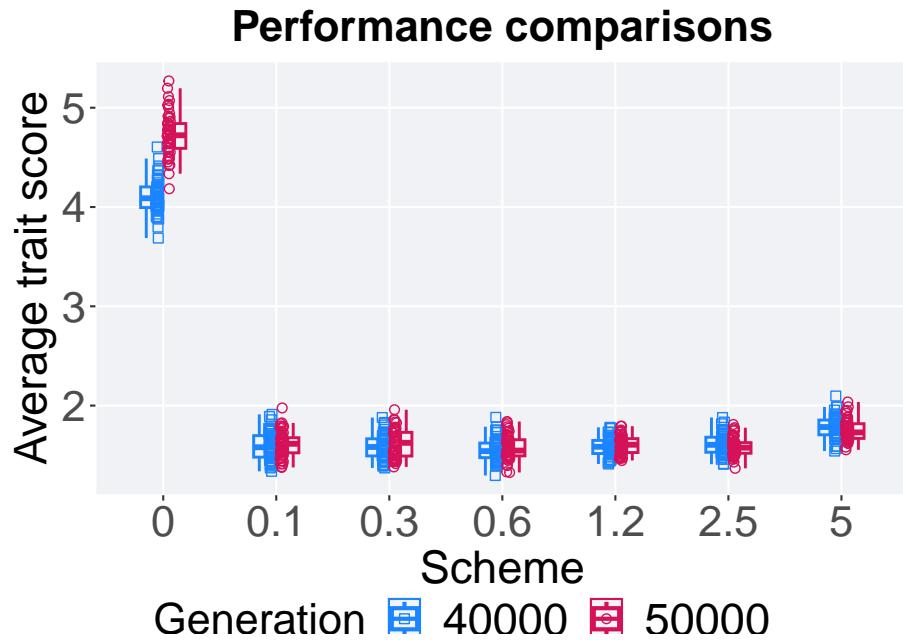
mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = Generation))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), nudge.y = 0)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = 0, y = 0))
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = 0, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  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
)

```



10.2.3.2 Stats

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

```
slices = filter(nds_ot_mvc, diagnostic == 'ordered_exploitation' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.
## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int>   <int>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

## 1 0      50000      50      0  4.18   4.72   4.73   5.27  0.250
## 2 0      40000      50      0  3.69   4.09   4.11   4.60  0.208
## 3 0.1    50000      50      0  1.38   1.62   1.61   1.98  0.152
## 4 0.1    40000      50      0  1.34   1.58   1.59   1.91  0.217
## 5 0.3    50000      50      0  1.38   1.63   1.61   1.96  0.239
## 6 0.3    40000      50      0  1.37   1.58   1.58   1.88  0.173
## 7 0.6    50000      50      0  1.33   1.55   1.58   1.84  0.163
## 8 0.6    40000      50      0  1.30   1.54   1.56   1.88  0.147
## 9 1.2    50000      50      0  1.45   1.61   1.60   1.79  0.140
## 10 1.2   40000      50      0  1.41   1.59   1.59   1.78  0.132
## 11 2.5    50000      50      0  1.37   1.58   1.58   1.81  0.107
## 12 2.5   40000      50      0  1.41   1.60   1.61   1.88  0.151
## 13 5     50000      50      0  1.55   1.73   1.75   2.04  0.148
## 14 5     40000      50      0  1.54   1.78   1.78   2.10  0.149

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 2462, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =
## W = 1355, p-value = 0.4713
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3
wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

```

```

## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 1413, p-value = 0.2626
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max
## W = 1392.5, p-value = 0.3276
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max
## W = 1375, p-value = 0.3907
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max
## W = 1059, p-value = 0.1891
## alternative hypothesis: true location shift is not equal to 0

Sigma 5.0

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma =  
## W = 980, p-value = 0.06319  
## alternative hypothesis: true location shift is not equal to 0
```

10.3 Contradictory objectives diagnostic

Here we present the results for **satisfactory trait coverage** and **activation gene coverage** found by each nondominated sorting sigma value replicate on the ordered exploitation diagnostic. Satisfactory trait coverage refers to the count of unique satisfied DIMENSIONALITY in the population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both coverage values fall between 0 and 100.

10.3.1 Satisfactory trait coverage

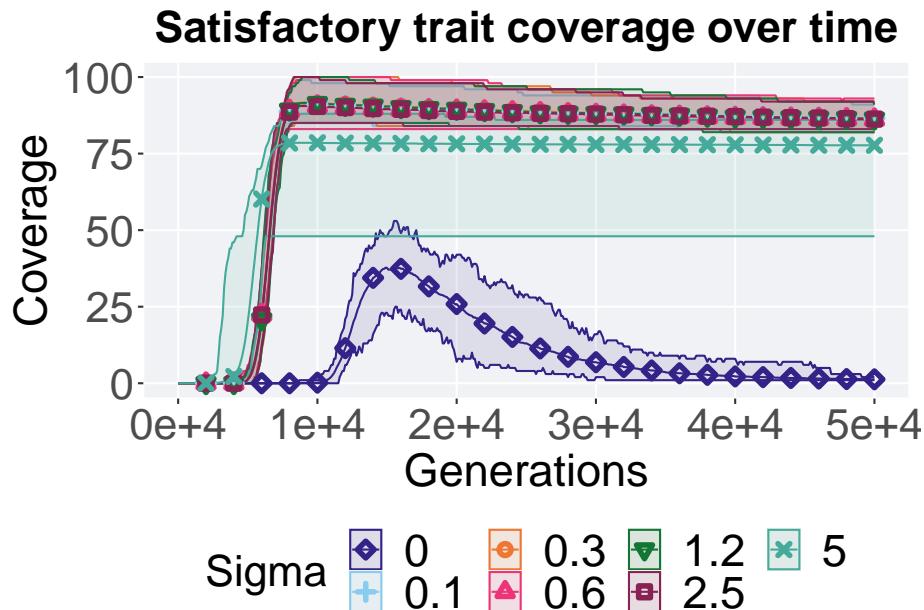
Satisfactory trait coverage analysis.

10.3.1.1 Coverage over time

Satisfactory trait coverage over time.

```
lines = filter(nds_ot, diagnostic == 'contradictory_objectives') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(pop_uni_obj),  
    mean = mean(pop_uni_obj),  
    max = max(pop_uni_obj)  
  )  
  
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## `.` argument.  
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +  
  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"  
  ) +  
  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")  
  ) +
```

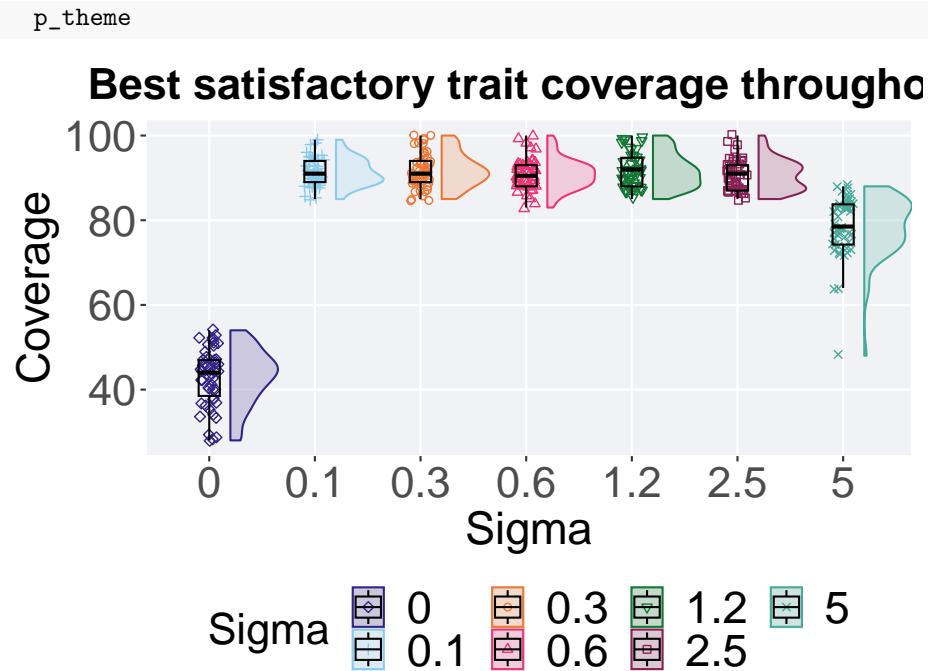
```
scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Satisfactory trait coverage over time") +
  p_theme
```



10.3.1.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
filter(nds_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = Sigma, y = val, color = Sigma, fill = Sigma, shape = Sigma)) +
  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"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best satisfactory trait coverage throughout") +
```



10.3.1.2.1 Stats

Summary statistics for the best satisfactory trait coverage.

```
coverage = filter(nds_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives')
  group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt  min median  mean  max    IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 0      50       0     28    44    42.8   54    8.5
## 2 0.1    50       0     85    91    91.4   99    5
## 3 0.3    50       0     85    91    91.8   100   5
## 4 0.6    50       0     83    90.5  90.9   100   5
## 5 1.2    50       0     85    92    91.8   100   6.75
```

```
## 6 2.5      50      0     85    91   90.7   100    6
## 7 5      50      0     48   78.5  78.6   88   9.5
```

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

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

```
##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 214.97, df = 6, p-value < 2.2e-16
```

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

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

```
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$val and coverage$Sigma
##
##      0      0.1 0.3 0.6 1.2 2.5
## 0.1 <2e-16 - - - - -
## 0.3 <2e-16 1 - - - - -
## 0.6 <2e-16 1 1 - - - -
## 1.2 <2e-16 1 1 1 - - -
## 2.5 <2e-16 1 1 1 1 - -
## 5 <2e-16 1 1 1 1 1
##
## P value adjustment method: bonferroni
```

10.3.1.3 End of 50,000 generations

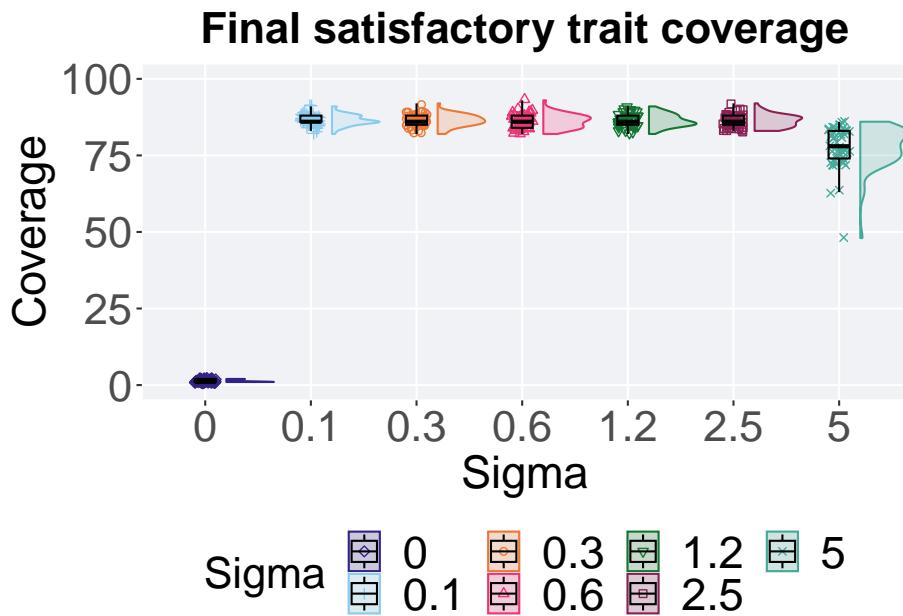
Satisfactory trait coverage in the population at the end of 50,000 generations.

```
filter(nds_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = pop_uni_obj, color = Sigma, fill = Sigma, shape = Sigma))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5)
  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)
  ) +
  scale_x_discrete()
```

```

  name="Sigma"
) +
scale_shape_manual(values=SHAPE) +
scale_colour_manual(values = cb_palette) +
scale_fill_manual(values = cb_palette) +
ggtitle("Final satisfactory trait coverage") +
p_theme

```



10.3.1.3.1 Stats

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

```

coverage = filter(nds_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, Sigma) %>%
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: 7 x 8

```

```
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <int> <dbl>
## 1 0      50     0     1     1 1.28     2     1
## 2 0.1    50     0    82    86 86.5    91     2
## 3 0.3    50     0    82    86 86.4    92     3
## 4 0.6    50     0    82    86 86.2    93     4
## 5 1.2    50     0    82    86 86.4    91     3
## 6 2.5    50     0    83    86 86.3    92     3
## 7 5      50     0    48    78 77.7    86     9
```

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

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

```
##
## Kruskal-Wallis rank sum test
##
## data: pop_uni_obj by Sigma
## Kruskal-Wallis chi-squared = 205.41, df = 6, p-value < 2.2e-16
```

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

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

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$pop_uni_obj and coverage$Sigma
##
##          0       0.1 0.3 0.6 1.2 2.5
## 0.1 <2e-16 -     -     -     -
## 0.3 <2e-16 1     -     -     -
## 0.6 <2e-16 1     1     -     -
## 1.2 <2e-16 1     1     1     -
## 2.5 <2e-16 1     1     1     -
## 5    <2e-16 1     1     1     1
##
## P value adjustment method: bonferroni
```

10.3.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the contradictory objectives diagnostic.

10.3.2.1 Coverage over time

Activation gene coverage over time.

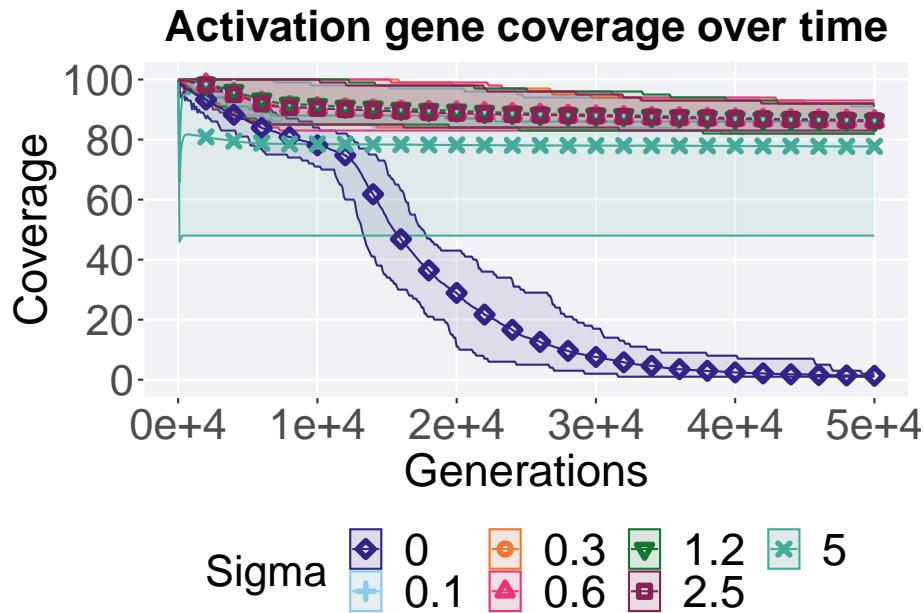
```

lines = filter(nds_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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(-1, 101),
    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

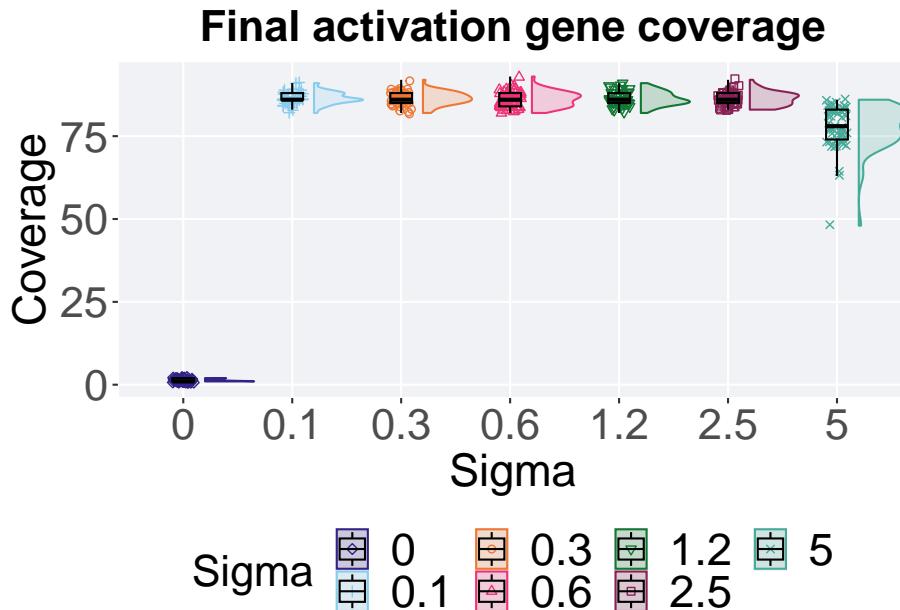
```



10.3.2.2 End of 50,000 generations

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

```
filter(nds_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = uni_str_pos, color = Sigma, fill = Sigma, shape = Sigma,
    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"
    ) +
    scale_x_discrete(
      name="Sigma"
    ) +
    scale_shape_manual(values=SHAPE) +
    scale_colour_manual(values = cb_palette) +
    scale_fill_manual(values = cb_palette) +
    ggtitle("Final activation gene coverage") +
    p_theme
```



10.3.2.2.1 Stats

Summary statistics for the best activation gene coverage in the population at the end of 50,000 generations.

```
coverage = filter(nds_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 0        50     0      1      1     1.3     2      1
## 2 0.1      50     0     82     86    86.5    91      2
## 3 0.3      50     0     82     86    86.4    92      3
## 4 0.6      50     0     82     86    86.2    93      4
## 5 1.2      50     0     82     86    86.4    91      3
## 6 2.5      50     0     83     86    86.3    92      3
```

```
## 7 5      50      0     48     78    77.7    86      9
```

Kruskal–Wallis test provides evidence of no statistical difference among activation gene coverage in the population at the end of 50,000 generations.

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

```
##  
## Kruskal-Wallis rank sum test  
##  
## data: uni_str_pos by Sigma  
## Kruskal-Wallis chi-squared = 205.39, df = 6, p-value < 2.2e-16
```

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

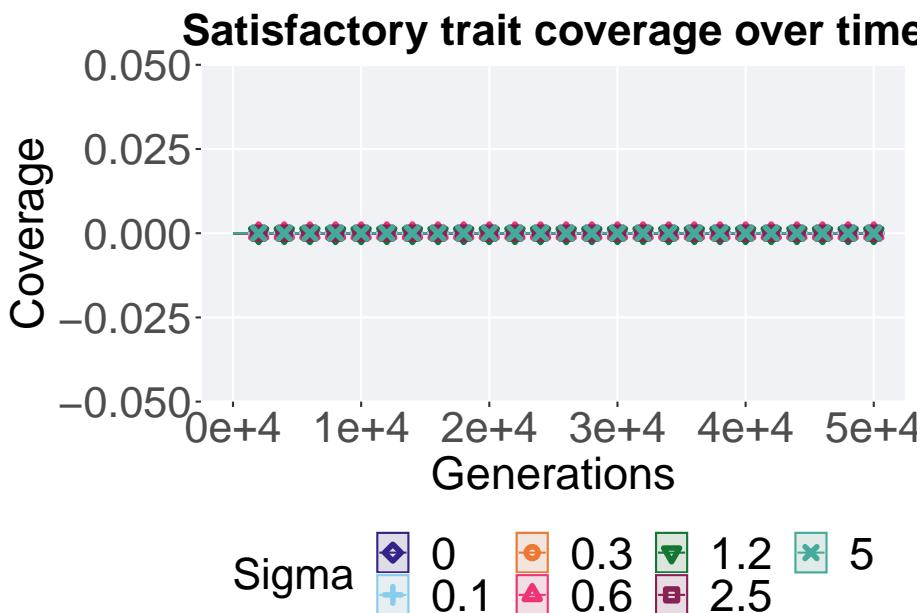
```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$Sigma, p.adjust.method =  
                      paired = FALSE, conf.int = FALSE, alternative = 'g')  
  
##  
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction  
##  
## data: coverage$uni_str_pos and coverage$Sigma  
##  
##      0      0.1 0.3 0.6 1.2 2.5  
## 0.1 <2e-16 - - - - -  
## 0.3 <2e-16 1 - - - -  
## 0.6 <2e-16 1 1 - - -  
## 1.2 <2e-16 1 1 1 - -  
## 2.5 <2e-16 1 1 1 1 -  
## 5   <2e-16 1 1 1 1 1  
##  
## P value adjustment method: bonferroni
```

10.3.3 Multi-valley crossing

10.3.3.1 Satisfactory trait coverage over time

```
lines = filter(nds_ot_mvc, diagnostic == 'contradictory_objectives') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(pop_uni_obj),  
    mean = mean(pop_uni_obj),  
    max = max(pop_uni_obj)  
  )  
  
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## ` .groups` argument.
```

```
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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 = 0.5) +
  scale_y_continuous(
    name="Coverage"
  ) +
  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
```



10.3.3.2 Satisfactory trait coverage comparison

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

```

# 80% and final generation comparison
end = filter(nds_ot_mvc, diagnostic == 'contradictory_objectives' & gen == 50000)
end$Generation <- factor(end$gen)

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

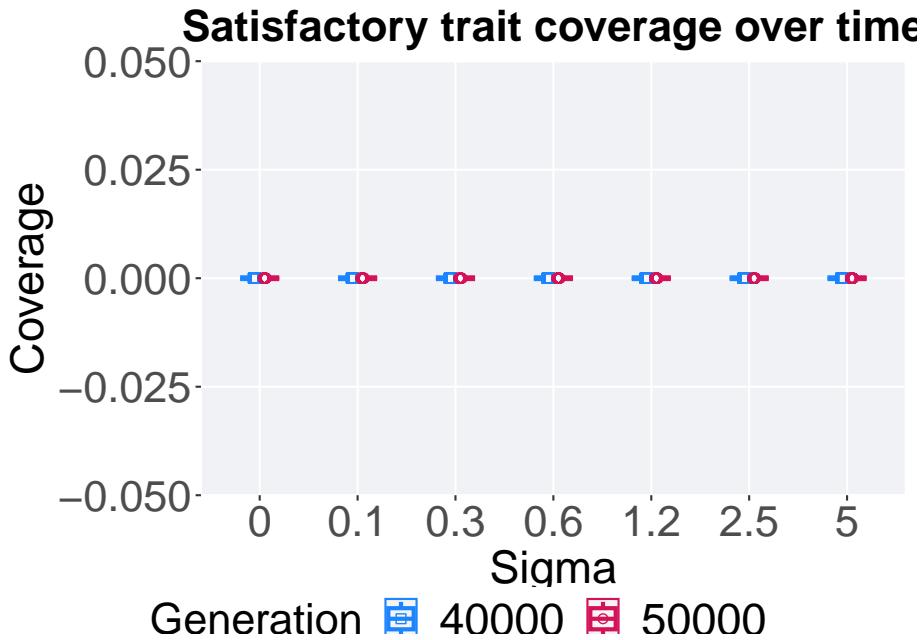
mvc_p = ggplot(mid, aes(x = Sigma, y=pop_uni_obj, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), lwd = 1, col = mvc_col[1])
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_uni_obj), col = mvc_col[2], position = position_nudge(x = 0))
  geom_boxplot(data = end, aes(x = Sigma, y=pop_uni_obj), position = position_nudge(x = 0))

  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
  )+
  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 over time") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



10.3.3.2.1 Stats

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

```
slices = filter(nds_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.` argument.
## # A tibble: 14 x 9
## # Groups:   Sigma [7]
##   Sigma Generation count na_cnt  min median  mean   max   IQR
##   <fct> <fct>     <int>  <int> <int>  <dbl> <dbl> <int> <dbl>
```

```

## 1 0      50000      50      0      0      0      0      0      0
## 2 0      40000      50      0      0      0      0      0      0
## 3 0.1    50000      50      0      0      0      0      0      0
## 4 0.1    40000      50      0      0      0      0      0      0
## 5 0.3    50000      50      0      0      0      0      0      0
## 6 0.3    40000      50      0      0      0      0      0      0
## 7 0.6    50000      50      0      0      0      0      0      0
## 8 0.6    40000      50      0      0      0      0      0      0
## 9 1.2    50000      50      0      0      0      0      0      0
## 10 1.2   40000      50      0      0      0      0      0      0
## 11 2.5   50000      50      0      0      0      0      0      0
## 12 2.5   40000      50      0      0      0      0      0      0
## 13 5     50000      50      0      0      0      0      0      0
## 14 5     40000      50      0      0      0      0      0      0

Sigma 0.0

wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1

wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.1 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

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

```

```
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.3 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 0.6 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 1.2 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_uni_obj and filter(slices, Sigma == 2.5 & Generation == 40000)$pop_uni_obj
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0

Sigma 5.0

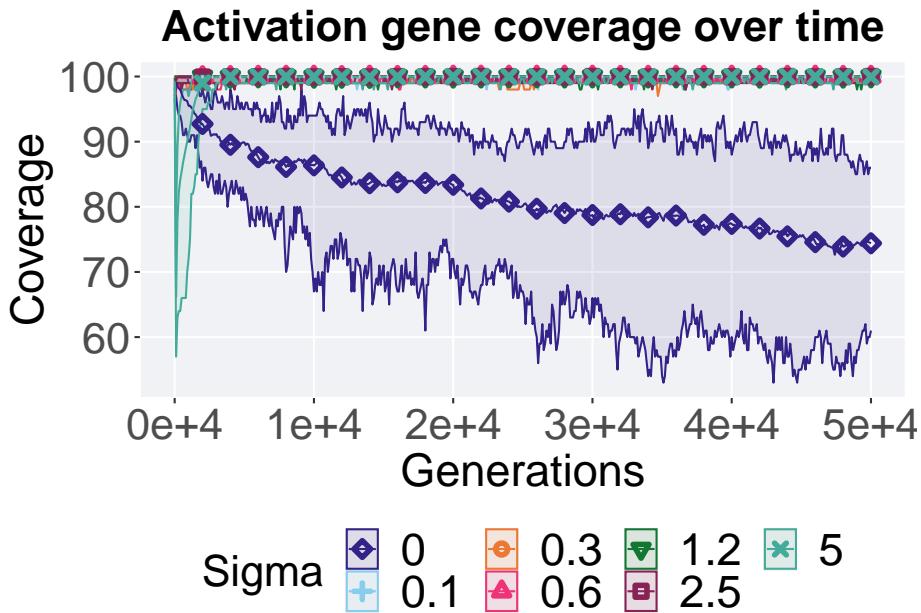
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_uni_obj,
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_uni_obj,
            alternative = 't')

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

```
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_uni_obj and filter(slices,  
## W = 1250, p-value = NA  
## alternative hypothesis: true location shift is not equal to 0
```

```
lines = filter(nds_ot_mvc, diagnostic == 'contradictory_objectives') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(uni_str_pos),  
    mean = mean(uni_str_pos),  
    max = max(uni_str_pos)  
  )
```

```
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## ` `.groups` argument.  
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +  
  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"  
  ) +  
  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
```



10.3.3.4 Activation gene coverage comparison

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

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

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

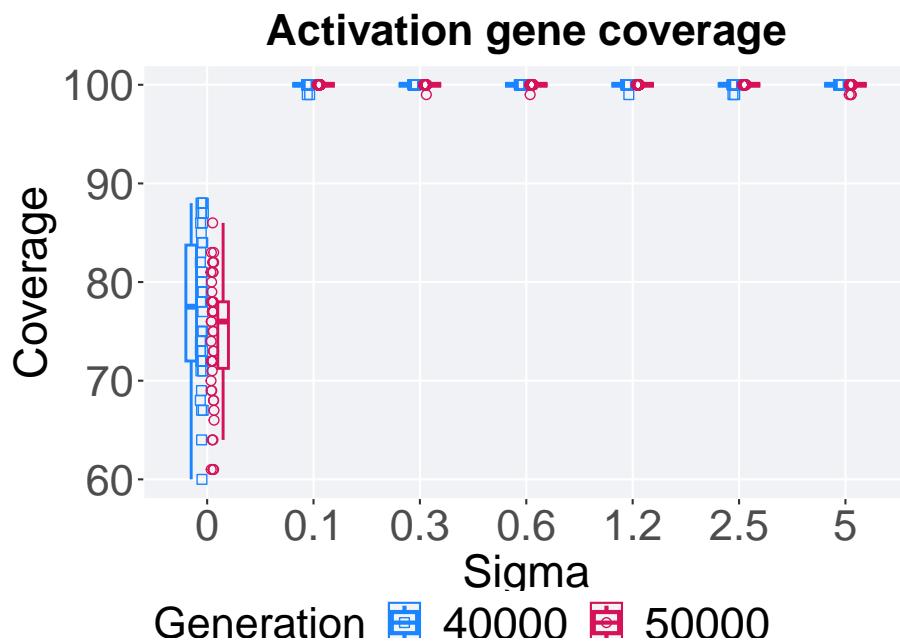
mvc_p = ggplot(mid, aes(x = Sigma, y=uni_str_pos, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 1.5) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
) +
  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") +
  theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)

```



10.3.3.4.1 Stats

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

```

slices = filter(nds_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 |
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <dbl> <dbl> <int> <dbl>
## 1 0     50000      50     0    61    76    74.4    86   6.75
## 2 0     40000      50     0    60    77.5   77.4    88  11.8
## 3 0.1   50000      50     0   100   100    100    100   0
## 4 0.1   40000      50     0    99   100    100.   100   0
## 5 0.3   50000      50     0    99   100    100.   100   0
## 6 0.3   40000      50     0   100   100    100    100   0
## 7 0.6   50000      50     0    99   100    100.   100   0
## 8 0.6   40000      50     0   100   100    100    100   0
## 9 1.2   50000      50     0   100   100    100    100   0
## 10 1.2  40000      50     0    99   100    100.   100   0
## 11 2.5   50000      50     0   100   100    100    100   0
## 12 2.5  40000      50     0    99   100    100.   100   0
## 13 5    50000      50     0    99   100    99.9   100   0
## 14 5    40000      50     0   100   100    100    100   0

Sigma 0.0

wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$uni_str_pos,
             alternative = 't')

## 
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =
## W = 969.5, p-value = 0.05315
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1

wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##

```

```

## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1300, p-value = 0.1594
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1225, p-value = 0.3271
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1225, p-value = 0.3271
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1275, p-value = 0.3271
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$uni_str_pos,
            alternative = 't')

```

```

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  

## W = 1300, p-value = 0.1594  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 5.0  

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$uni_str_pos,  

            y = filter(slices, Sigma == 5.0 & Generation == 40000)$uni_str_pos,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  

## W = 1175, p-value = 0.08218  

## alternative hypothesis: true location shift is not equal to 0

```

10.4 Multi-path exploration results

Here we present the results for **best performances** and **activation gene coverage** found by each nondominated sorting sigma value replicate on the multi-path exploration diagnostic. Best performance found refers to the largest average trait score found in a given population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both values fall between 0 and 100.

10.4.1 Performance

Here we analyze the performances for each parameter replicate on the multi-path exploration diagnostic.

10.4.1.1 Performance over time

Performance over time.

```

lines = filter(nds_ot, diagnostic == 'multipath_exploration') %>%  

  group_by(Sigma, gen) %>%  

  dplyr::summarise(  

    min = min(pop_fit_max),  

    mean = mean(pop_fit_max),  

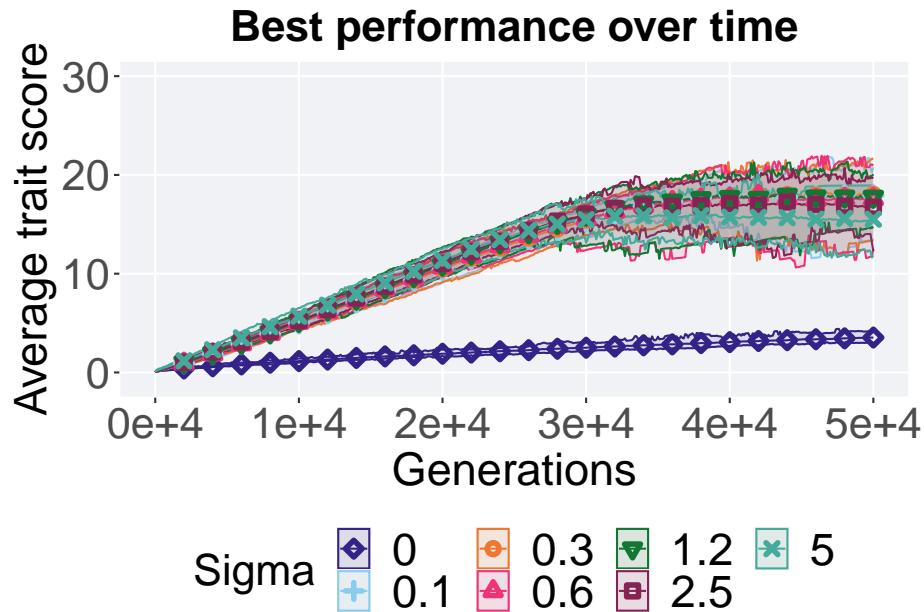
    max = max(pop_fit_max)
  )  

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

## `.` argument.

```

```
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma,
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2),
  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(-1, 30)
  ) +
  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("Best performance over time") +
  p_theme
```

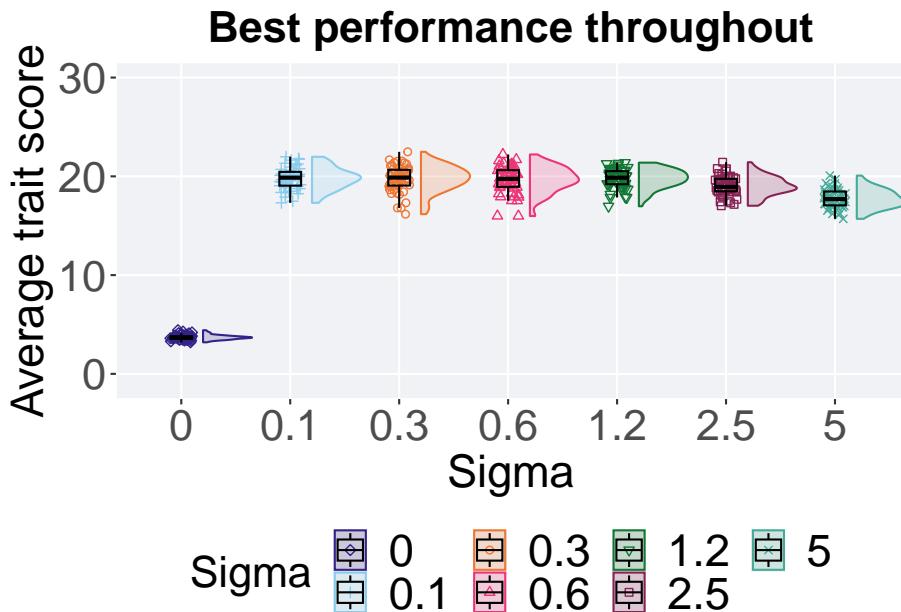


10.4.1.2 Best performance throughout

Here we plot the performance of the best performing solution found throughout 50,000 generations.

```

filter(nds_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
  ggplot(., aes(x = Sigma, y = val / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma)) +
    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, 30)
    ) +
    scale_x_discrete(
      name = "Sigma"
    ) +
    scale_shape_manual(values = SHAPE) +
    scale_colour_manual(values = cb_palette) +
    scale_fill_manual(values = cb_palette) +
    ggtitle("Best performance throughout") +
    p_theme
  
```



10.4.1.2.1 Stats

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

```

performance = filter(nds_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration')
group_by(performance, Sigma) %>%
  
```

```
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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0      50     0  3.20  3.68  3.71  4.42  0.248
## 2 0.1    50     0 17.3   19.9  19.8  22.0  1.39
## 3 0.3    50     0 16.2   19.9  19.8  22.5  1.59
## 4 0.6    50     0 16.0   19.7  19.6  22.2  1.69
## 5 1.2    50     0 17.0   19.9  19.8  21.4  1.33
## 6 2.5    50     0 17.0   18.9  19.0  21.4  1.27
## 7 5      50     0 15.7   17.7  17.7  20.1  1.40
```

Kruskal–Wallis test provides evidence of statistical differences among the best performing solution found throughout 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: val by Sigma
## Kruskal-Wallis chi-squared = 192.67, df = 6, p-value < 2.2e-16

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performing solution found throughout 50,000 generations.

pairwise.wilcox.test(x = performance$val, g = performance$Sigma, p.adjust.method = "bo
                           paired = FALSE, conf.int = FALSE, alternative = 't')

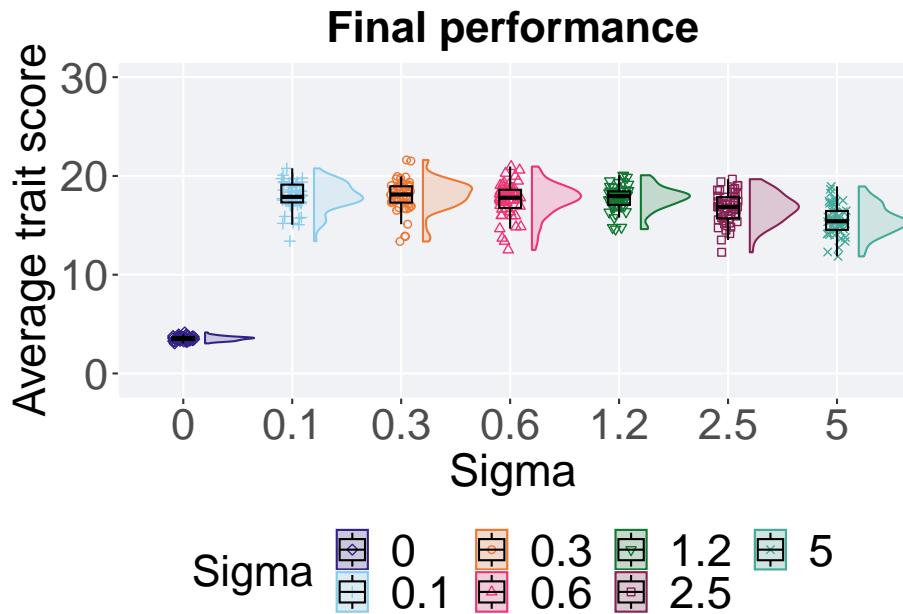
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$Sigma
##
##      0      0.1     0.3     0.6     1.2     2.5
## 0.1 < 2e-16 -      -      -      -      -
## 0.3 < 2e-16 1.0000 -      -      -      -
## 0.6 < 2e-16 1.0000 1.0000 -      -      -
## 1.2 < 2e-16 1.0000 1.0000 1.0000 -      -
```

```
## 2.5 < 2e-16 0.0194 0.0199 0.1236 0.0058 -
## 5   < 2e-16 1.3e-11 7.8e-10 2.6e-09 8.8e-12 4.7e-07
##
## P value adjustment method: bonferroni
```

10.4.1.3 End of 50,000 generations

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

```
filter(nds_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = pop_fit_max / DIMENSIONALITY, color = Sigma, fill = Sigma, shape = Sigma))
  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, 30)
  ) +
  scale_x_discrete(
    name = "Sigma"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final performance") +
  p_theme
```



10.4.1.3.1 Stats

Summary statistics for the best performance in the population at the end of 50,000 generations.

```
performance = filter(nds_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(performance, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt   min median   mean   max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0         50     0  3.05   3.57   3.54  4.15  0.275
## 2 0.1       50     0 13.4    17.9   17.9  20.8   1.80
## 3 0.3       50     0 13.4    18.1   18.0  21.6   1.65
## 4 0.6       50     0 12.5    17.8   17.6  21.0   1.84
## 5 1.2       50     0 14.6    18.0   17.8  20.1   1.37
## 6 2.5       50     0 12.3    16.9   16.7  19.7   2.15
## 7 5         50     0 11.9    15.4   15.5  18.9   1.91
```

Kruskal–Wallis test provides evidence of statistical differences among best performance in the population at the end of 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: pop_fit_max by Sigma
## Kruskal-Wallis chi-squared = 182.39, df = 6, p-value < 2.2e-16
```

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

```
pairwise.wilcox.test(x = performance$pop_fit_max, g = performance$Sigma , p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$pop_fit_max and performance$Sigma
```

```
##  
##      0      0.1      0.3      0.6      1.2      2.5  
## 0.1 < 2e-16 -      -      -      -      -  
## 0.3 < 2e-16 1.00000 -      -      -      -  
## 0.6 < 2e-16 1.00000 1.00000 -      -      -  
## 1.2 < 2e-16 1.00000 1.00000 1.00000 -      -  
## 2.5 < 2e-16 0.00221 0.00091 0.12360 0.01237 -  
## 5     < 2e-16 4.1e-09 4.2e-09 8.7e-07 6.9e-09 0.00443  
##  
## P value adjustment method: bonferroni
```

10.4.2 Activation gene coverage

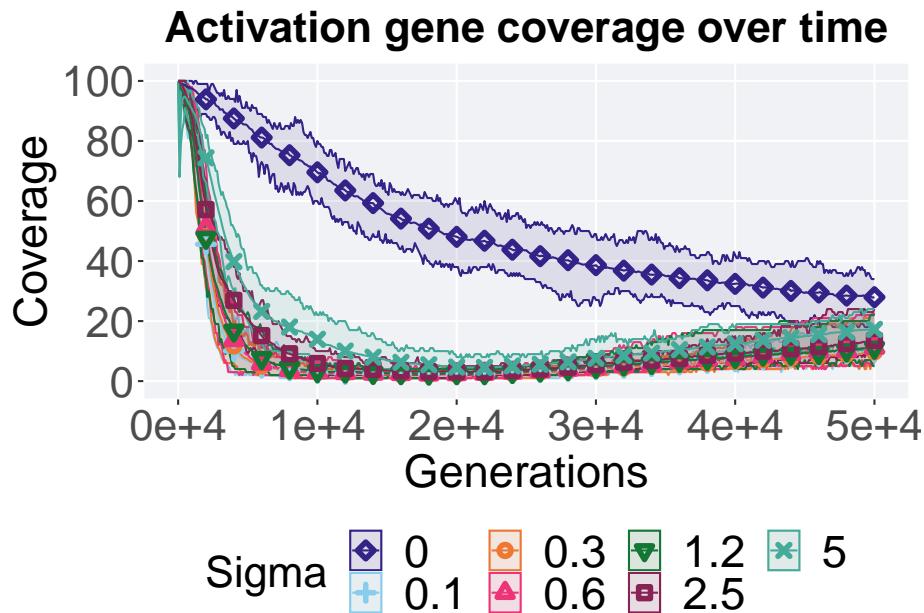
Here we analyze the activation gene coverage for each parameter replicate on the multi-path exploration diagnostic.

10.4.2.1 Coverage over time

Activation gene coverage over time.

```
lines = filter(nds_ot, diagnostic == 'multipath_exploration') %>%  
  group_by(Sigma, gen) %>%  
  dplyr::summarise(  
    min = min(uni_str_pos),  
    mean = mean(uni_str_pos),  
    max = max(uni_str_pos)  
  )  
  
## `summarise()` has grouped output by 'Sigma'. You can override using the  
## `.groups` argument.  
ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +  
  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(-1, 101),  
    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
```

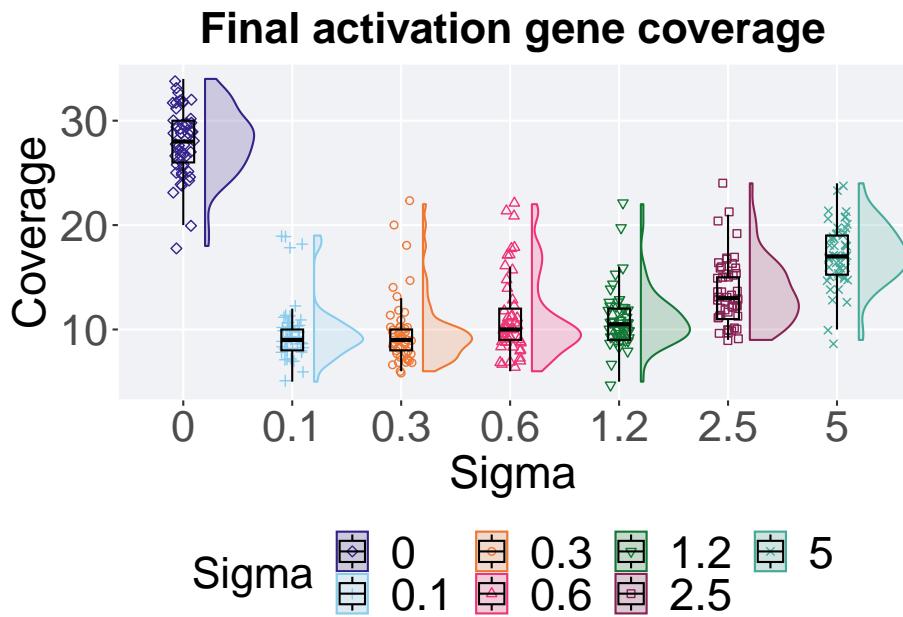


10.4.2.2 End of 50,000 generations

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

```
filter(nds_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = Sigma, y = uni_str_pos, color = Sigma, fill = Sigma, shape = Sigma)) +
  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"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
```

```
ggtile("Final activation gene coverage") +
p_theme
```



10.4.2.2.1 Stats

Summary statistics for the activation gene coverage in the population at the end of 50,000 generations.

```
coverage = filter(nds_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(coverage, Sigma) %>%
  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: 7 x 8
##   Sigma count na_cnt  min median  mean  max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 0        50      0     18    28    28.0    34     4
## 2 0.1      50      0      5     9     9.76    19     2
## 3 0.3      50      0      6     9     9.76    22     2
```

```
## 4 0.6      50     0     6    10   11.2     22   3
## 5 1.2      50     0     5   10.5 11.0     22   3
## 6 2.5      50     0     9    13   13.6     24   4
## 7 5        50     0     9    17   17.2     24 3.75
```

Kruskal–Wallis test provides evidence of statistical differences among activation gene coverage in the population at the end of 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: uni_str_pos by Sigma
## Kruskal-Wallis chi-squared = 218.65, df = 6, p-value < 2.2e-16
```

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

```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$Sigma , p.adjust.method =
  paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$uni_str_pos and coverage$Sigma
##
##          0       0.1      0.3      0.6      1.2      2.5
## 0.1 < 2e-16 -       -       -       -       -
## 0.3 < 2e-16 1.00000 -       -       -       -
## 0.6 < 2e-16 1.00000 0.65720 -       -       -
## 1.2 < 2e-16 0.01207 0.01118 1.00000 -       -
## 2.5 2.3e-16 6.0e-09 1.3e-08 0.00084 0.00019 -
## 5   8.2e-16 1.7e-12 1.4e-12 4.4e-09 4.3e-12 8.0e-07
##
## P value adjustment method: bonferroni
```

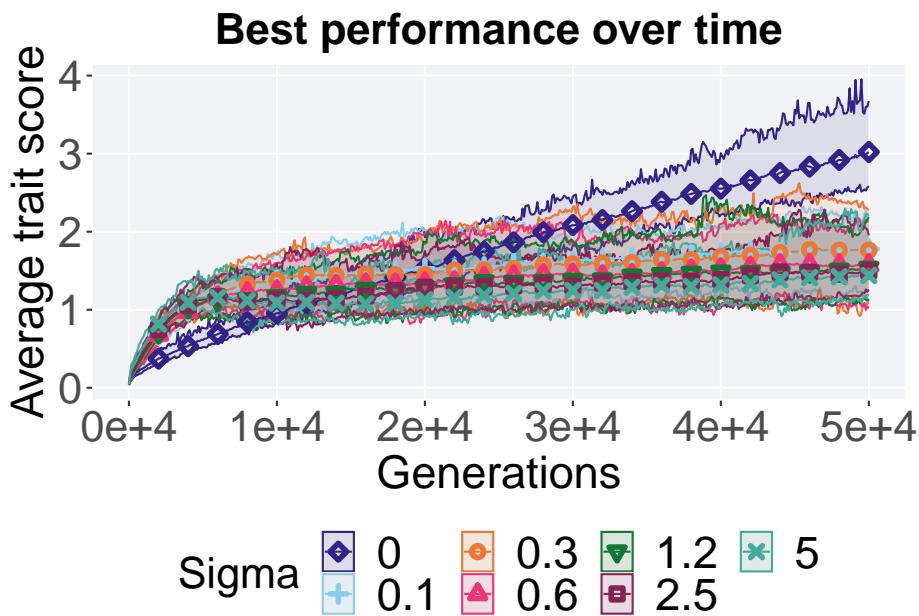
10.4.3 Multi-valley crossing

10.4.3.1 Performance over time

```
lines = filter(nds_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )
```

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

```
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma))
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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 = 1)
  scale_y_continuous(
    name="Average trait score"
  ) +
  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("Best performance over time") +
  p_theme
```



10.4.3.2 Performance comparisons

```

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

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

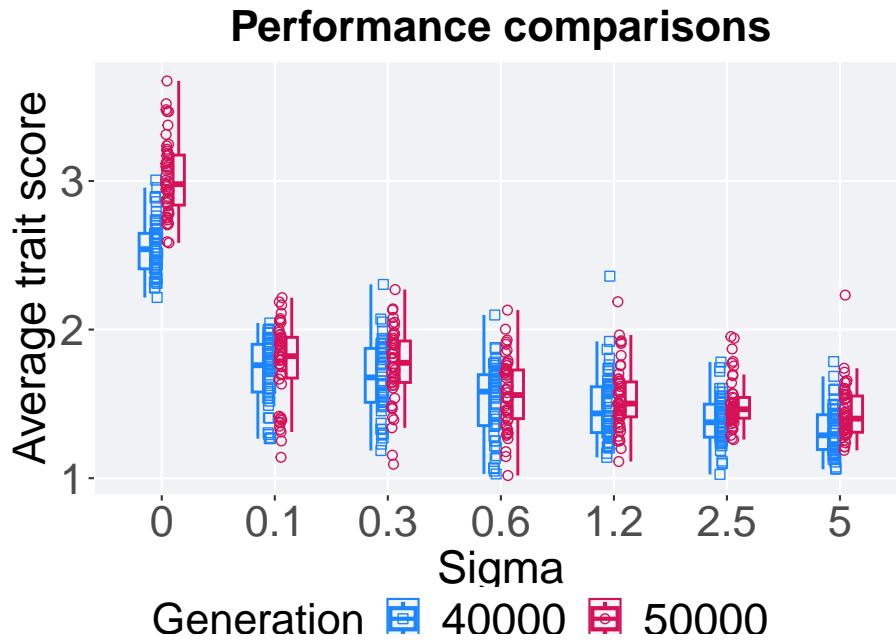
mvc_p = ggplot(mid, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY, group = Sigma, shape = mvc_col[1]))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15))
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2])
  geom_boxplot(data = end, aes(x = Sigma, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15))

  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="Sigma"
  ) +
  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
)

```



10.4.3.2.1 Stats

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

```
# 80% and final generation comparison
slices = filter(nds_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.`groups` argument.

## # A tibble: 14 x 9
## # Groups:   Sigma [7]
##   Sigma Generation count  na_cnt   min  median   mean   max     IQR
##   <dbl>      <dbl> <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

##   <fct> <fct>     <int>  <int> <dbl>  <dbl> <dbl> <dbl>
## 1 0    50000      50     0  2.58  2.98  3.02  3.67 0.337
## 2 0    40000      50     0  2.22  2.54  2.55  3.01 0.238
## 3 0.1  50000      50     0  1.14  1.82  1.78  2.21 0.274
## 4 0.1  40000      50     0  1.27  1.76  1.72  2.04 0.321
## 5 0.3  50000      50     0  1.09  1.78  1.76  2.27 0.279
## 6 0.3  40000      50     0  1.19  1.68  1.68  2.30 0.363
## 7 0.6  50000      50     0  1.02  1.56  1.57  2.13 0.327
## 8 0.6  40000      50     0  1.03  1.58  1.52  2.10 0.342
## 9 1.2  50000      50     0  1.11  1.50  1.53  2.19 0.235
## 10 1.2 40000      50     0  1.14  1.44  1.49  2.36 0.308
## 11 2.5  50000      50     0  1.26  1.46  1.50  1.95 0.140
## 12 2.5  40000      50     0  1.03  1.38  1.39  1.78 0.222
## 13 5    50000      50     0  1.19  1.40  1.44  2.23 0.244
## 14 5    40000      50     0  1.06  1.29  1.32  1.78 0.234

Sigma 0.0

wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.0 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0 & Generation == 40000)$pop_fit_max
## W = 2343, p-value = 5.016e-14
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1

wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$pop_fit_max and filter(slices, Sigma == 0.1 & Generation == 40000)$pop_fit_max
## W = 1465, p-value = 0.1392
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$pop_fit_max,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```

##  

## data: filter(slices, Sigma == 0.3 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  

## W = 1509, p-value = 0.07474  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 0.6  

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 0.6 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 0.6 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  

## W = 1383, p-value = 0.361  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 1.2  

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 1.2 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 1.2 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  

## W = 1447, p-value = 0.1755  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 2.5  

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 2.5 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##  

## Wilcoxon rank sum test with continuity correction  

##  

## data: filter(slices, Sigma == 2.5 & Generation == 50000)$pop_fit_max and filter(slices, Sigma  

## W = 1704, p-value = 0.00177  

## alternative hypothesis: true location shift is not equal to 0  

Sigma 5.0  

wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$pop_fit_max,  

            y = filter(slices, Sigma == 5.0 & Generation == 40000)$pop_fit_max,  

            alternative = 't')  

##
```

```

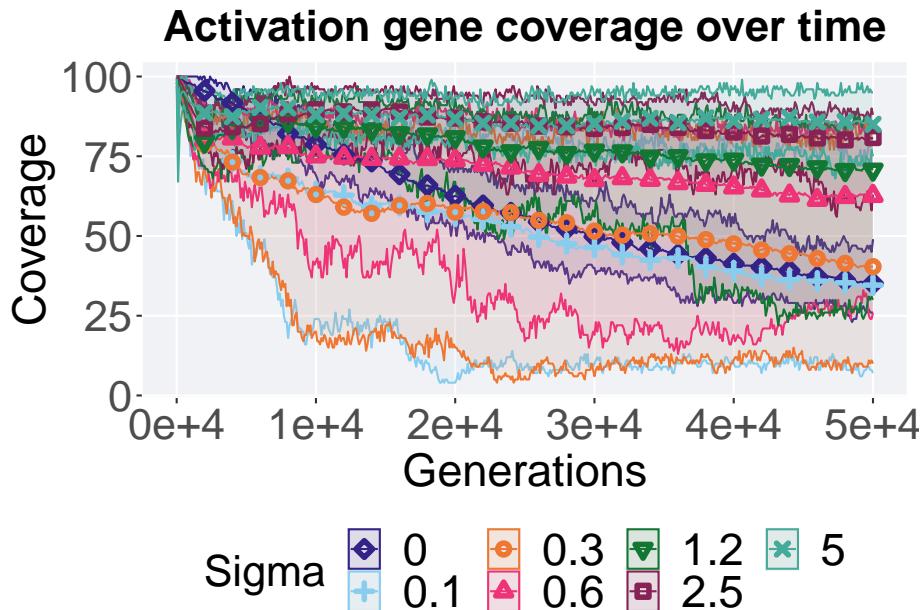
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 5 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1736, p-value = 0.0008171
## alternative hypothesis: true location shift is not equal to 0

lines = filter(nds_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(Sigma, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = Sigma, fill = Sigma, color = Sigma, shape = Sigma)) +
  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"
  ) +
  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

```



10.4.3.4 Activation gene coverage comparison

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

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

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

mvc_p = ggplot(mid, aes(x = Sigma, y=uni_str_pos, group = Sigma, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), fill = mvc_col[1])
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = Sigma, y=uni_str_pos), col = mvc_col[2], position = position_jitternudge(jitter.width = .03, nudge.x = 0.15), fill = mvc_col[2])
  geom_boxplot(data = end, aes(x = Sigma, y=uni_str_pos), position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

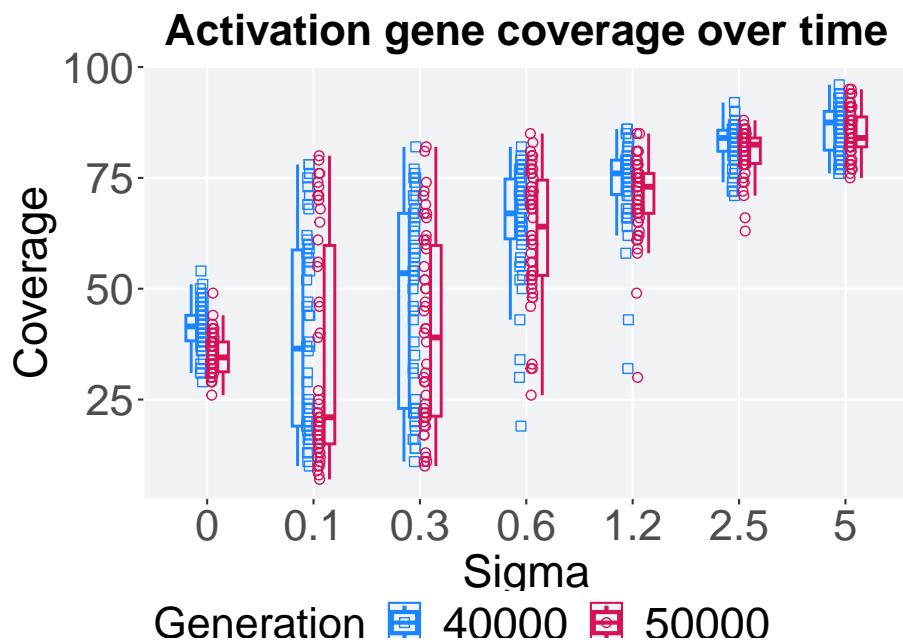
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="Sigma"
) +
  scale_shape_manual(values=c(0,1))+
```

```

scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
p_theme

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

```



10.4.3.4.1 Stats

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

```

slices = filter(nds_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen ==
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(Sigma, 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 'Sigma'. You can override using the
## `.groups` argument.

## # A tibble: 14 x 9
## # Groups: Sigma [7]
##   Sigma Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>     <int> <int> <dbl> <dbl> <int> <dbl>
## 1 0      50000      50    0    26   34.5  35.1    49  6.75
## 2 0      40000      50    0    29   41.5  41.3    54  5.75
## 3 0.1    50000      50    0     7   21    34.6    80 44.8
## 4 0.1    40000      50    0    10   36.5  39.1    78 39.8
## 5 0.3    50000      50    0    10   39    40.4    82 38.5
## 6 0.3    40000      50    0    11   53.5  47.5    82 44
## 7 0.6    50000      50    0    26   64    62.6    85 21.5
## 8 0.6    40000      50    0    19   67    65.3    82 13.5
## 9 1.2    50000      50    0    30   73    70.7    85  9
## 10 1.2   40000      50    0    32   76    73.9    86 7.75
## 11 2.5    50000      50    0    63   82.5  80.9    88 5.75
## 12 2.5   40000      50    0    71   84    82.6    92 4.75
## 13 5     50000      50    0    75   84    85      95 6.75
## 14 5     40000      50    0    76   87.5  86.1    96 8.75

Sigma 0.0
wilcox.test(x = filter(slices, Sigma == 0.0 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.0 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =
## W = 452.5, p-value = 3.703e-08
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.1
wilcox.test(x = filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos,
             y = filter(slices, Sigma == 0.1 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##

```

```

## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.1 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1060.5, p-value = 0.1924
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.3

wilcox.test(x = filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.3 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.3 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1008, p-value = 0.09584
## alternative hypothesis: true location shift is not equal to 0

Sigma 0.6

wilcox.test(x = filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 0.6 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 0.6 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1101, p-value = 0.3057
## alternative hypothesis: true location shift is not equal to 0

Sigma 1.2

wilcox.test(x = filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 1.2 & Generation == 40000)$uni_str_pos,
            alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, Sigma == 1.2 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 910, p-value = 0.01909
## alternative hypothesis: true location shift is not equal to 0

Sigma 2.5

wilcox.test(x = filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos,
            y = filter(slices, Sigma == 2.5 & Generation == 40000)$uni_str_pos,
            alternative = 't')

```

```
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 2.5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  
## W = 998.5, p-value = 0.0823  
## alternative hypothesis: true location shift is not equal to 0  
Sigma 5.0  
wilcox.test(x = filter(slices, Sigma == 5.0 & Generation == 50000)$uni_str_pos,  
            y = filter(slices, Sigma == 5.0 & Generation == 40000)$uni_str_pos,  
            alternative = 't')  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: filter(slices, Sigma == 5 & Generation == 50000)$uni_str_pos and filter(slices, Sigma =  
## W = 1096.5, p-value = 0.2906  
## alternative hypothesis: true location shift is not equal to 0
```


Chapter 11

Novelty Search

We present the results from our parameter sweep on novelty search. 50 replicates are conducted for each K-nearest neighbors parameter value explored.

```
library(ggplot2)
library(cowplot)
library(dplyr)
library(Pupillometry)
```

11.1 Exploitation rate results

Here we present the results for best performances found by each novelty search K value 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 and 100.

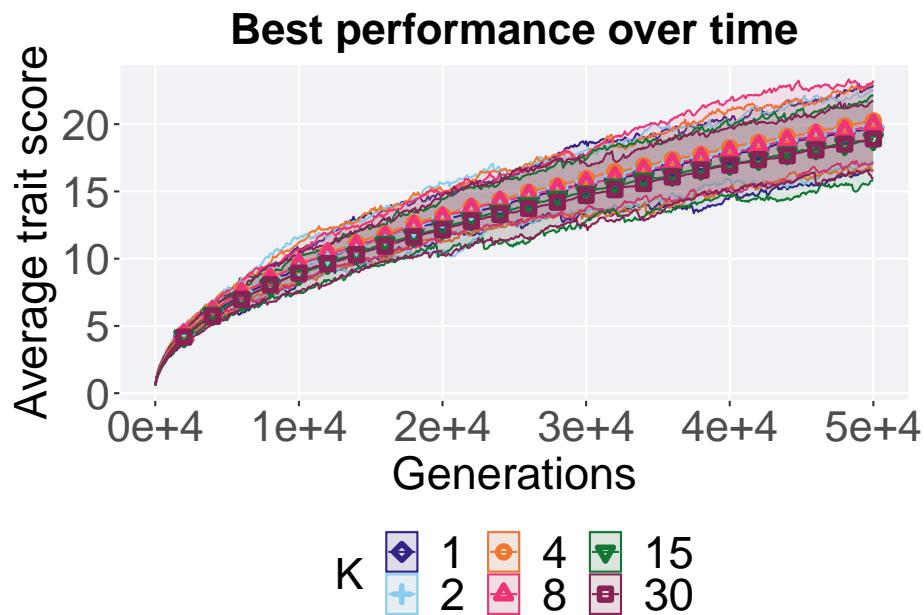
11.1.1 Performance over time

Performance over time.

```
lines = filter(nov_ot, diagnostic == 'exploitation_rate') %>%
  group_by(K, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

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

```
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = K, fill = K, shape = K))
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2)
  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"
  ) +
  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("Best performance over time") +
  p_theme
```



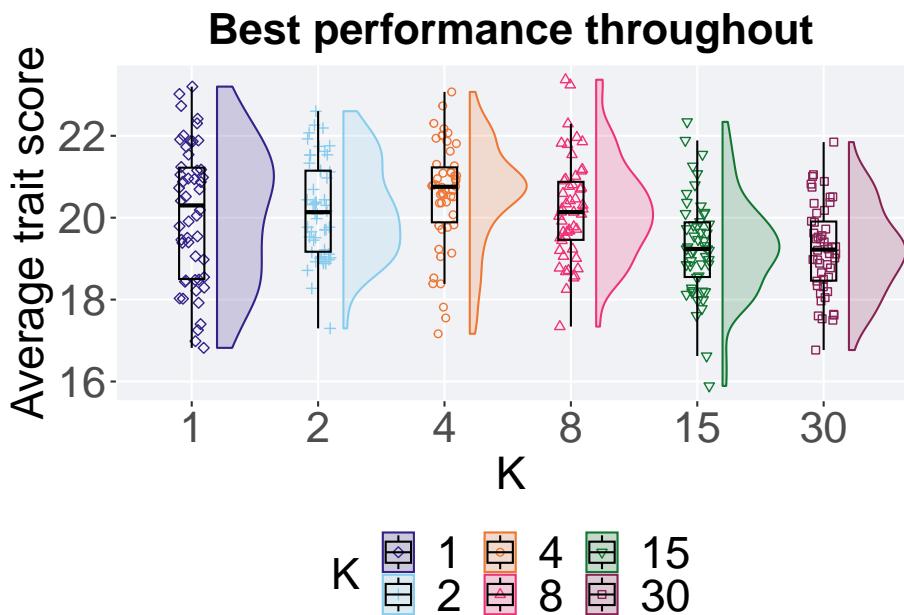
11.1.2 Best performance throughout

The best performance found throughout 50,000 generations.

```

filter(nov_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate') %>%
ggplot(., aes(x = K, y = val / DIMENSIONALITY, color = K, fill = K, shape = K)) +
  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"
  ) +
  scale_x_discrete(
    name="K"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best performance throughout") +
  p_theme

```



11.1.2.1 Stats

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

```

performance = filter(nov_best, col == 'pop_fit_max' & diagnostic == 'exploitation_rate');
group_by(performance, K) %>%
  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: 6 x 8
##   K     count na_cnt    min median   mean    max    IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1      50     0  16.8  20.3  20.1  23.2  2.72
## 2 2      50     0  17.3  20.1  20.2  22.6  1.98
## 3 4      50     0  17.2  20.8  20.6  23.1  1.34
## 4 8      50     0  17.3  20.1  20.2  23.4  1.42
## 5 15     50     0  15.9  19.2  19.3  22.3  1.34
## 6 30     50     0  16.8  19.2  19.2  21.8  1.44

```

Kruskal–Wallis test provides evidence of significant differences among K values on the best performance found throughout 50,000 generations

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

```

##
## Kruskal-Wallis rank sum test
##
## data: val by K
## Kruskal-Wallis chi-squared = 41.239, df = 5, p-value = 8.394e-08

```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performance found throughout 50,000 generations.

```

pairwise.wilcox.test(x = performance$val, g = performance$K, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$K
##
##   1     2     4     8     15
## 2 1.0000 -     -     -     -
## 4 1.0000 1.0000 -     -     -
## 8 1.0000 1.0000 0.5941 -     -
## 15 0.1905 0.0068 3.9e-05 0.0072 -
## 30 0.0777 0.0023 6.2e-06 0.0025 1.0000
##
```

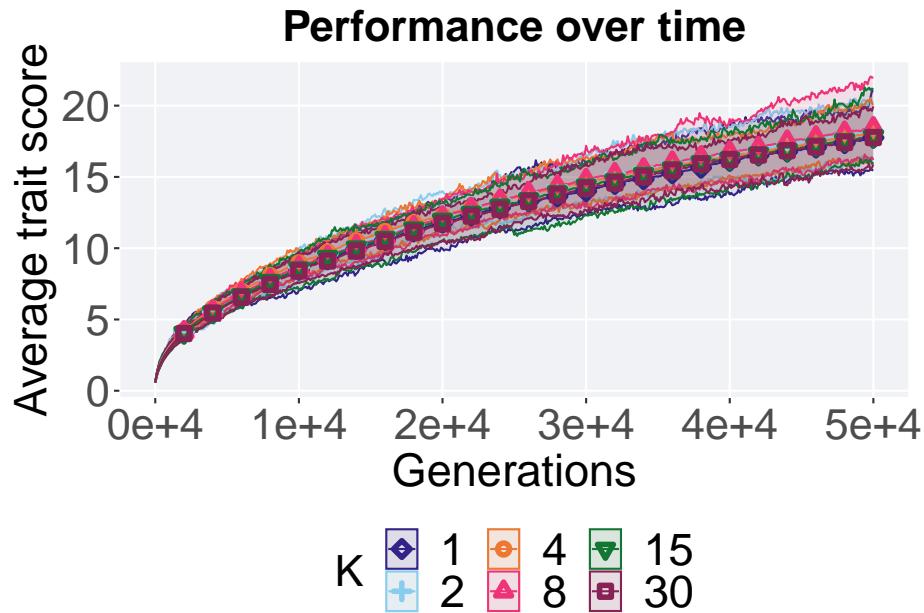
```
## P value adjustment method: bonferroni
```

11.1.3 Multi-valley crossing

```
# data for lines and shading on plots
lines = filter(nov_ot_mvc, diagnostic == 'exploitation_rate') %>%
  group_by(K, 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 'K'. You can override using the `groups` argument.

ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +
  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="Average trait score"
  ) +
  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
```



11.1.3.2 Performance comparison

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

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

mid = filter(nov_ot_mvc, diagnostic == 'exploitation_rate' & gen == 40000)
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = K, y=pop_fit_max / DIMENSIONALITY, group = K, shape = Generation))
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge_x = -.15), size = 100)
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1])

  geom_point(data = end, aes(x = K, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], size = 100)
  geom_boxplot(data = end, aes(x = K, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[2])

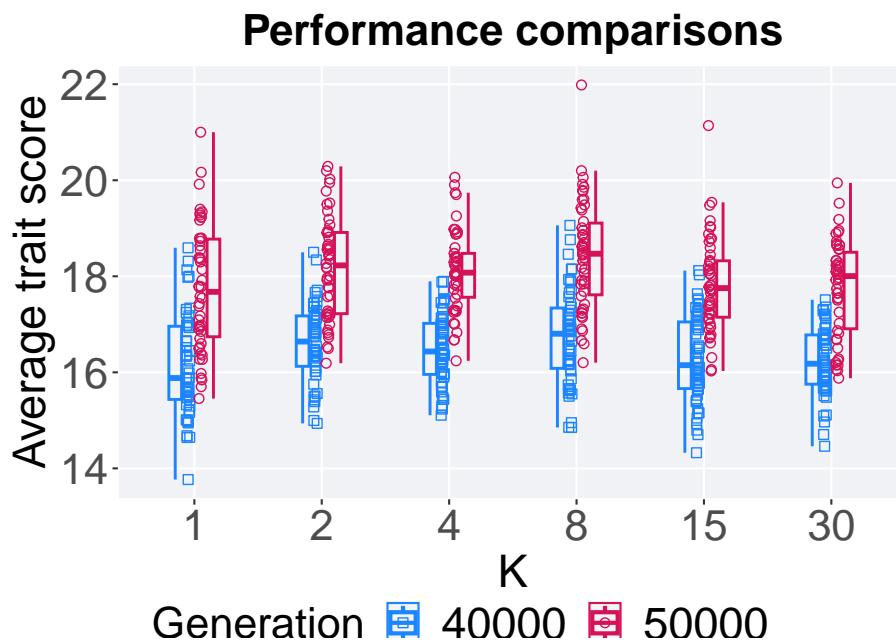
  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="K"
  ) +
  scale_shape_manual(values=c(0,1)) +
```

```

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
)

```



11.1.3.3 Stats

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

```

slices = filter(nov_ot_mvc, diagnostic == 'exploitation_rate' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(K, 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 'K'. You can override using the `groups` argument.

## # A tibble: 12 x 9
## # Groups: K [6]
##   K     Generation count na_cnt   min median   mean   max   IQR
##   <fct>    <fct>     <int>   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1      50000       50     0  15.5  17.7  17.7  21.0 2.03
## 2 1      40000       50     0  13.8  15.9  16.1  18.6 1.52
## 3 2      50000       50     0  16.2  18.2  18.2  20.3 1.69
## 4 2      40000       50     0  14.9  16.6  16.6  18.5 1.05
## 5 4      50000       50     0  16.2  18.1  18.1  20.1 0.914
## 6 4      40000       50     0  15.1  16.4  16.5  17.9 1.06
## 7 8      50000       50     0  16.2  18.5  18.4  22.0 1.49
## 8 8      40000       50     0  14.9  16.8  16.7  19.1 1.26
## 9 15     50000       50     0  16.0  17.8  17.8  21.1 1.17
## 10 15    40000       50     0  14.3  16.1  16.3  18.1 1.39
## 11 30     50000       50     0  15.9  18.0  17.8  19.9 1.59
## 12 30    40000       50     0  14.5  16.2  16.2  17.5 1.03

K 1

wilcox.test(x = filter(slices, K == 1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 1 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 2073, p-value = 1.427e-08
## alternative hypothesis: true location shift is not equal to 0

K 2

wilcox.test(x = filter(slices, K == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

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

```

```

## data: filter(slices, K == 2 & Generation == 50000)$pop_fit_max and filter(slices, K == 2 & Ge
## W = 2191, p-value = 8.954e-11
## alternative hypothesis: true location shift is not equal to 0
K 4
wilcox.test(x = filter(slices, K == 4 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 4 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 4 & Generation == 50000)$pop_fit_max and filter(slices, K == 4 & Ge
## W = 2315, p-value = 2.16e-13
## alternative hypothesis: true location shift is not equal to 0
K 8
wilcox.test(x = filter(slices, K == 8 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 8 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 8 & Generation == 50000)$pop_fit_max and filter(slices, K == 8 & Ge
## W = 2178, p-value = 1.616e-10
## alternative hypothesis: true location shift is not equal to 0
K 15
wilcox.test(x = filter(slices, K == 15 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 15 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 15 & Generation == 50000)$pop_fit_max and filter(slices, K == 15 &
## W = 2196, p-value = 7.119e-11
## alternative hypothesis: true location shift is not equal to 0
K 30
wilcox.test(x = filter(slices, K == 30 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 30 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```
##  
## data: filter(slices, K == 30 & Generation == 50000)$pop_fit_max and filter(slices,  
## W = 2200, p-value = 5.922e-11  
## alternative hypothesis: true location shift is not equal to 0
```

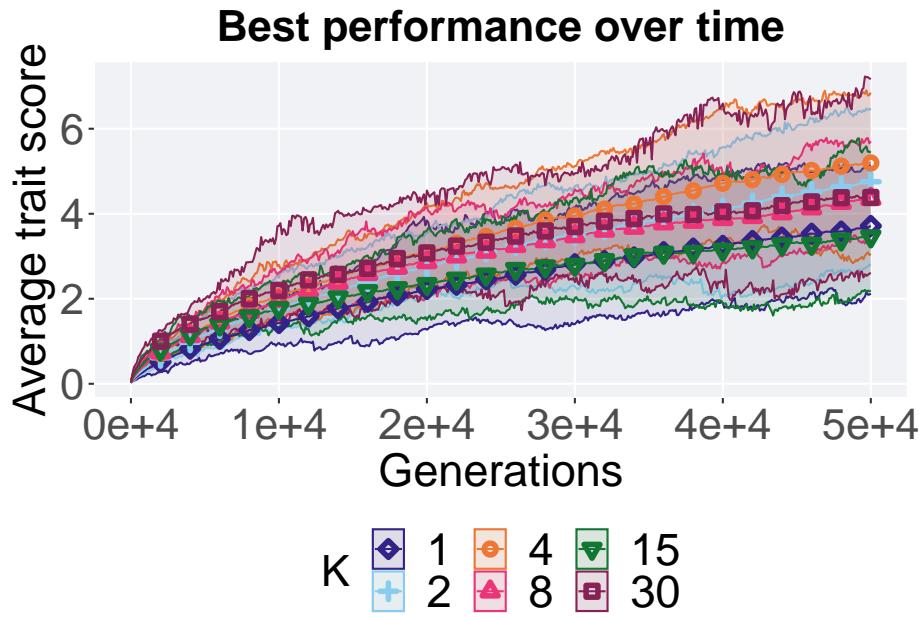
11.2 Ordered exploitation results

Here we present the results for **best performances** found by each novelty search K value 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 and 100.

11.2.1 Performance over time

Performance over time.

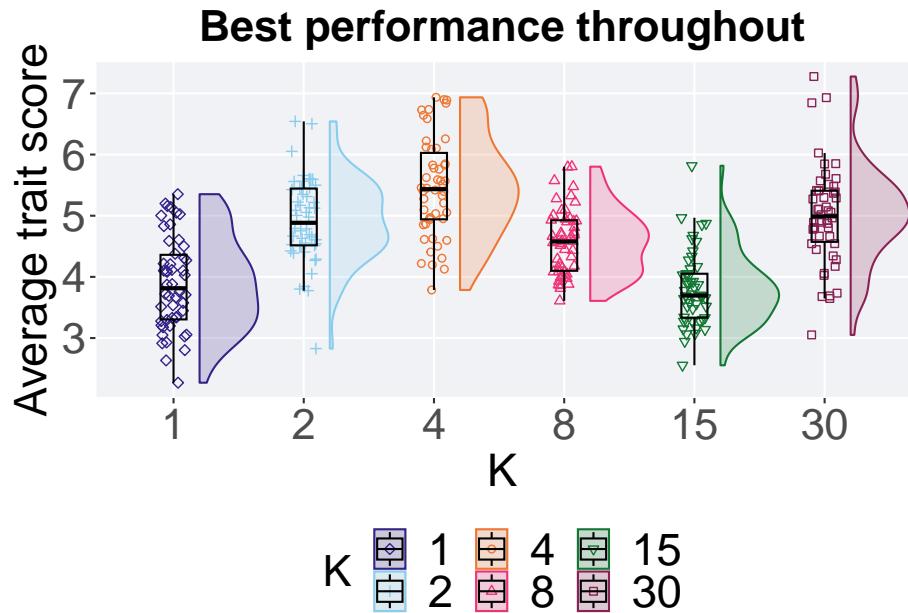
```
lines = filter(nov_ot, diagnostic == 'ordered_exploitation') %>%  
  group_by(K, gen) %>%  
  dplyr::summarise(  
    min = min(pop_fit_max),  
    mean = mean(pop_fit_max),  
    max = max(pop_fit_max)  
  )  
  
## `summarise()` has grouped output by 'K'. You can override using the `.groups`  
## argument.  
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = K, color = K, shape = 0)) +  
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2) +  
  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"  
  ) +  
  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("Best performance over time") +  
  p_theme
```



11.2.2 Best performance throughout

The best performance found throughout 50,000 generations.

```
filter(nov_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploitation') %>%
  ggplot(., aes(x = K, y = val / DIMENSIONALITY, color = K, fill = K, shape = K)) +
  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"
  ) +
  scale_x_discrete(
    name="K"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best performance throughout") +
  p_theme
```



11.2.2.1 Stats

Summary statistics about the best performance found.

```
performance = filter(nov_best, col == 'pop_fit_max' & diagnostic == 'ordered_exploit')
group_by(performance, K) %>%
  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: 6 x 8
##   K     count na_cnt   min  median   mean   max    IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1      50     0  2.27   3.81   3.89   5.35  1.06
## 2 2      50     0  2.83   4.88   4.92   6.54  0.928
## 3 4      50     0  3.79   5.43   5.46   6.94  1.09
## 4 8      50     0  3.61   4.58   4.58   5.80  0.826
## 5 15     50     0  2.55   3.70   3.80   5.82  0.718
## 6 30     50     0  3.05   4.99   5.00   7.28  0.835
```

Kruskal–Wallis test provides evidence of statistical differences for the best performance found in the population throughout 50,000 generations.

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

##
## Kruskal-Wallis rank sum test
##
## data: val by K
## Kruskal-Wallis chi-squared = 127.68, df = 5, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performance found in the population throughout 50,000 generations.

```
pairwise.wilcox.test(x = performance$val, g = performance$K, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')
```

```
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$K
##
##    1      2      4      8      15
## 2 5.1e-08 -      -      -      -
## 4 6.3e-12 0.02655 -      -      -
## 8 0.00017 0.11567 1.1e-06 -      -
## 15 1.00000 2.9e-10 1.5e-13 5.5e-08 -
## 30 8.4e-08 1.00000 0.10010 0.03273 4.9e-10
##
## P value adjustment method: bonferroni
```

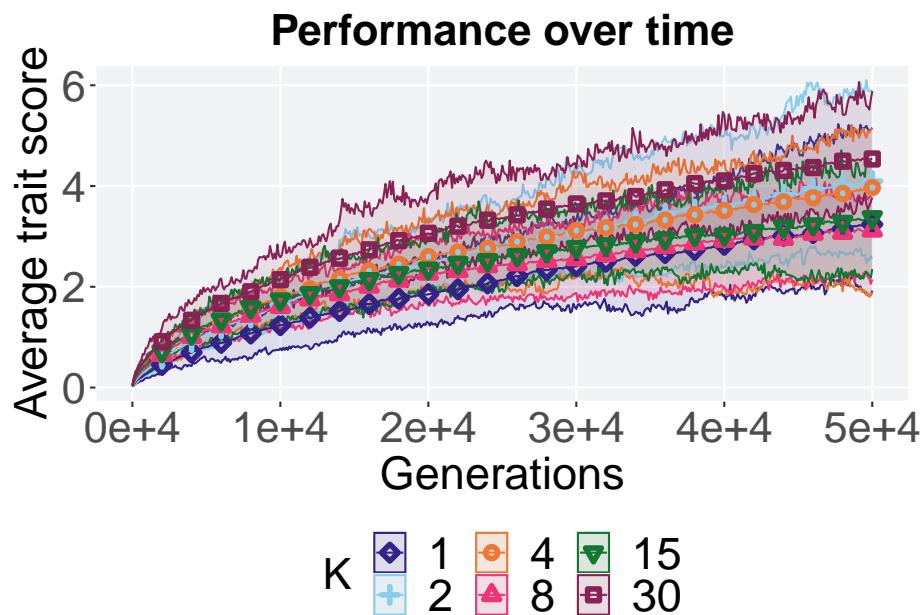
11.2.3 Multi-valley crossing

11.2.3.1 Performance over time

```
# data for lines and shading on plots
lines = filter(nov_ot_mvc, diagnostic == 'ordered_exploitation') %>%
  group_by(K, 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 'K'. You can override using the `.groups` argument.
```

```
ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +
  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"
  ) +
  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
```



11.2.3.2 Performance comparison

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

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

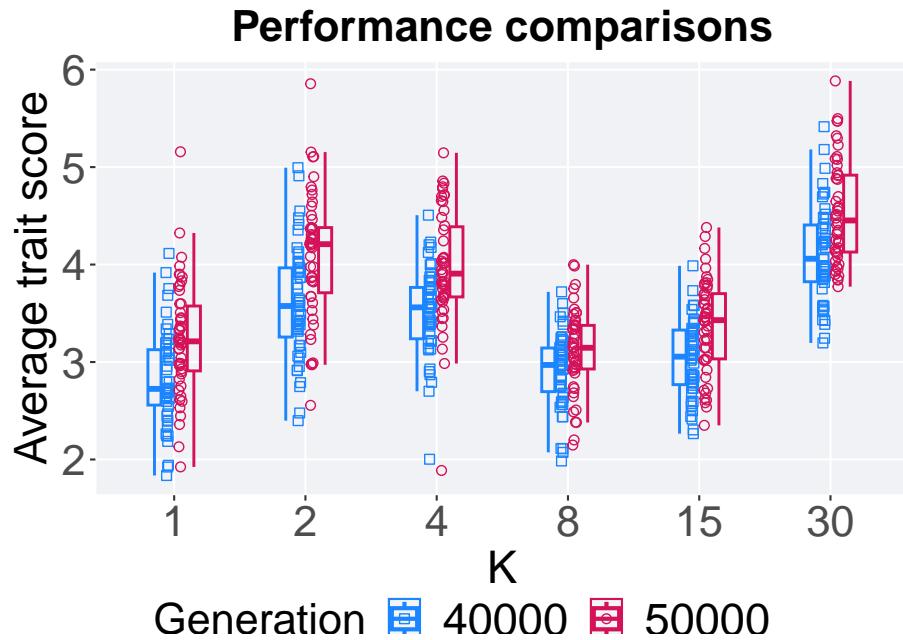
mid = filter(nov_ot_mvc, diagnostic == 'ordered_exploitation' & gen == 40000)
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = K, y=pop_fit_max / DIMENSIONALITY, group = K, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), col = mvc_col[1], fill = mvc_col[1])
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = K, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2], position = position_nudge(x = 0, y = 0))
  geom_boxplot(data = end, aes(x = K, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = 0, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="K"
  ) +
  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
)
```



11.2.3.3 Stats

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

```
slices = filter(nov_ot_mvc, diagnostic == 'ordered_exploitation' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(K, 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 'K'. You can override using the `.groups` argument.

## # A tibble: 12 x 9
## # Groups:   K [6]
##   K     Generation count  na_cnt   min  median   mean   max    IQR
##   <fct> <fct>     <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

##  1 1      50000      50      0  1.92   3.21   3.24   5.16 0.666
##  2 1      40000      50      0  1.84   2.72   2.83   4.11 0.569
##  3 2      50000      50      0  2.56   4.21   4.10   5.86 0.669
##  4 2      40000      50      0  2.40   3.57   3.61   4.99 0.710
##  5 4      50000      50      0  1.89   3.91   3.97   5.15 0.720
##  6 4      40000      50      0  2.00   3.56   3.51   4.51 0.527
##  7 8      50000      50      0  2.15   3.15   3.12   4.00 0.448
##  8 8      40000      50      0  1.99   2.97   2.92   3.72 0.448
##  9 15     50000      50      0  2.35   3.43   3.38   4.38 0.670
## 10 15    40000      50      0  2.27   3.06   3.03   3.99 0.560
## 11 30     50000      50      0  3.77   4.45   4.54   5.88 0.789
## 12 30    40000      50      0  3.20   4.06   4.10   5.41 0.582

K 1
wilcox.test(x = filter(slices, K == 1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 1 & Generation == 50000)$pop_fit_max and filter(slices, K == 1 & Ge
## W = 1779, p-value = 0.0002691
## alternative hypothesis: true location shift is not equal to 0

K 2
wilcox.test(x = filter(slices, K == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 2 & Generation == 50000)$pop_fit_max and filter(slices, K == 2 & Ge
## W = 1809, p-value = 0.000118
## alternative hypothesis: true location shift is not equal to 0

K 4
wilcox.test(x = filter(slices, K == 4 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 4 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 4 & Generation == 50000)$pop_fit_max and filter(slices, K == 4 & Ge
## W = 1889, p-value = 1.074e-05

```

```

## alternative hypothesis: true location shift is not equal to 0
K 8
wilcox.test(x = filter(slices, K == 8 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 8 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 8 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1608, p-value = 0.01372
## alternative hypothesis: true location shift is not equal to 0

K 15
wilcox.test(x = filter(slices, K == 15 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 15 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 15 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1789, p-value = 0.0002054
## alternative hypothesis: true location shift is not equal to 0

K 30
wilcox.test(x = filter(slices, K == 30 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 30 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 30 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1799, p-value = 0.000156
## alternative hypothesis: true location shift is not equal to 0

```

11.3 Contradictory objectives diagnostic

Here we present the results for **satisfactory trait coverage** and **activation gene coverage** found by each novelty search K value replicate on the contradictory objectives diagnostic. Satisfactory trait coverage refers to the count of unique satisfied traits in the population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both coverage values fall between 0 and 100.

11.3.1 Satisfactory trait coverage

Here we analyze the satisfactory trait coverage for each parameter replicate on the contradictory objectives diagnostic.

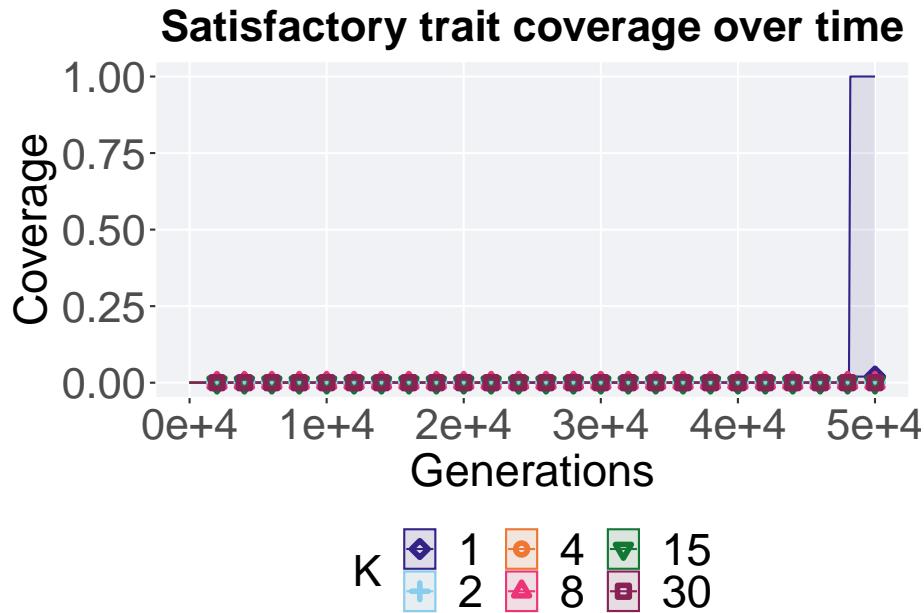
11.3.1.1 Coverage over time

Satisfactory trait coverage over time.

```
lines = filter(nov_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(K, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )

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

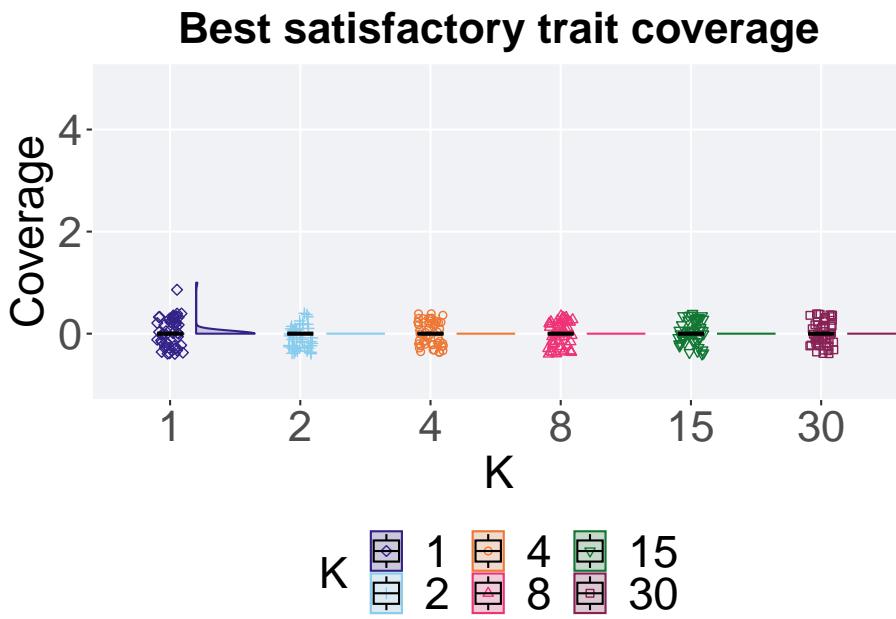
ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage"
  ) +
  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
```



11.3.1.2 Best coverage throughout

Best satisfactory trait coverage throughout 50,000 generations.

```
filter(nov_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives') %>%
  ggplot(., aes(x = K, y = val, color = K, fill = K, shape = K)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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(-1, 5)
  ) +
  scale_x_discrete(
    name = "K"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Best satisfactory trait coverage") +
  p_theme
```



11.3.1.2.1 Stats

Summary statistics for the best satisfactory trait coverage throughout 50,000 generations.

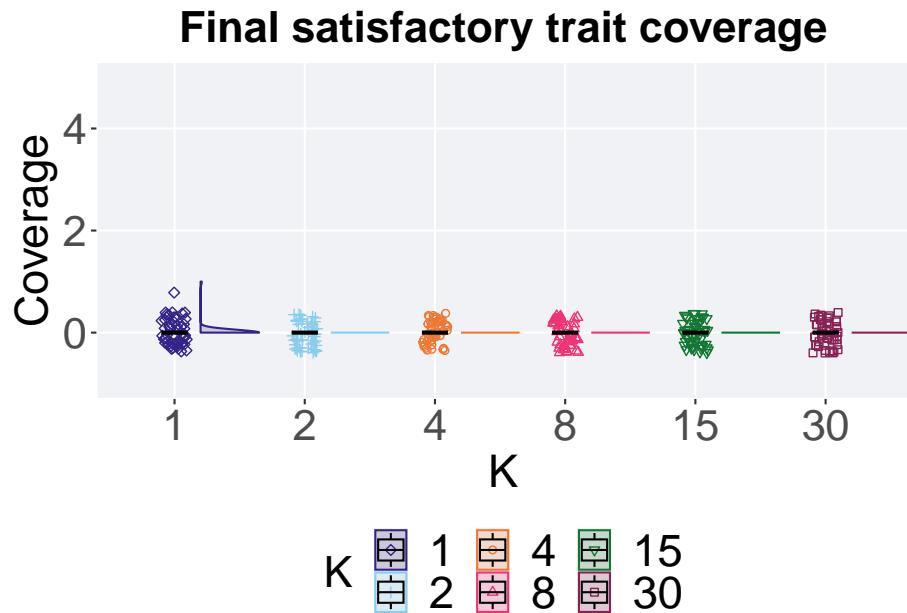
```
coverage = filter(nov_best, col == 'pop_uni_obj' & diagnostic == 'contradictory_objectives')
group_by(coverage, K) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(val)),
    min = min(val, na.rm = TRUE),
    median = median(val, na.rm = TRUE),
    mean = mean(val, na.rm = TRUE),
    max = max(val, na.rm = TRUE),
    IQR = IQR(val, na.rm = TRUE)
  )

## # A tibble: 6 x 8
##   K      count na_cnt  min median  mean  max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1       50     0     0     0    0.02    1     0
## 2 2       50     0     0     0    0     0     0     0
## 3 4       50     0     0     0    0     0     0     0
## 4 8       50     0     0     0    0     0     0     0
## 5 15      50     0     0     0    0     0     0     0
## 6 30      50     0     0     0    0     0     0     0
```

11.3.1.3 End of 50,000 generations

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

```
filter(nov_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
  ggplot(., aes(x = K, y = pop_uni_obj, color = K, fill = K, shape = K)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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(-1, 5)
  ) +
  scale_x_discrete(
    name = "K"
  ) +
  scale_shape_manual(values = SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final satisfactory trait coverage") +
  p_theme
```



11.3.1.3.1 Stats

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

```

coverage = filter(nov_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, K) %>%
  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: 6 x 8
##   K     count na_cnt   min median   mean   max   IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl>
## 1 1       50      0     0     0  0.02     1     0
## 2 2       50      0     0     0     0     0     0
## 3 4       50      0     0     0     0     0     0
## 4 8       50      0     0     0     0     0     0
## 5 15      50      0     0     0     0     0     0
## 6 30      50      0     0     0     0     0     0

```

11.3.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the contradictory objectives diagnostic.

11.3.2.1 Coverage over time

Activation gene coverage over time.

```

lines = filter(nov_ot, diagnostic == 'contradictory_objectives') %>%
  group_by(K, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +
  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()

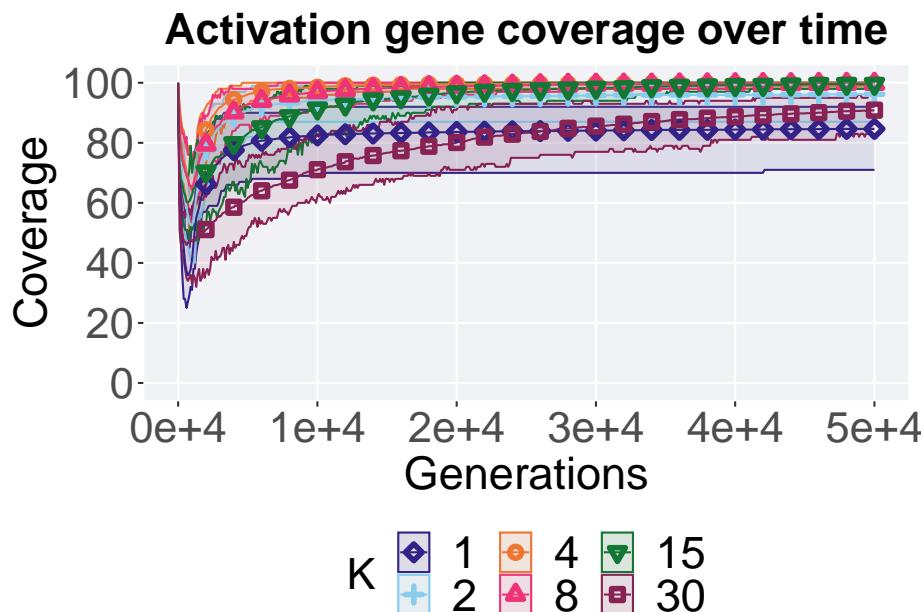
```

```

name="Coverage",
limits=c(-1, 101),
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

```



11.3.2.2 End of 50,000 generations

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

```

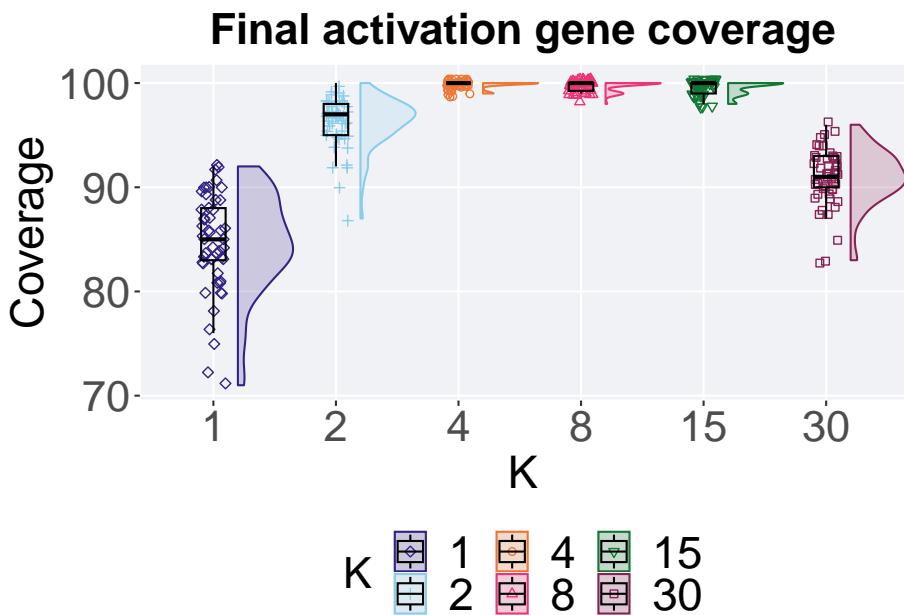
filter(nov_ot, diagnostic == 'contradictory_objectives' & gen == 50000) %>%
ggplot(., aes(x = K, y = uni_str_pos, color = K, fill = K, shape = K)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5)

```

```

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"
  ) +
  scale_x_discrete(
    name="K"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final activation gene coverage") +
  p_theme

```



11.3.2.2.1 Stats

Summary statistics for the activation gene coverage in the population at the end of 50,000 generations.

```

coverage = filter(nov_ot, diagnostic == 'contradictory_objectives' & gen == 50000)
group_by(coverage, K) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(uni_str_pos)),
    min = min(uni_str_pos, na.rm = TRUE),
    max = max(uni_str_pos, na.rm = TRUE),
    mean = mean(uni_str_pos, na.rm = TRUE),
    median = median(uni_str_pos, na.rm = TRUE),
    sd = sd(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: 6 x 8
##   K     count na_cnt  min median  mean    max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 1      50     0    71    85  84.6    92     5
## 2 2      50     0    87    97  96.2   100     3
## 3 4      50     0    99   100  99.8   100     0
## 4 8      50     0    98   100  99.7   100  0.75
## 5 15     50     0    98   100  99.6   100     1
## 6 30     50     0    83    91  90.9    96     3

```

Kruskal–Wallis test provides evidence of statistical differences among activation gene coverage in the population at the end of 50,000 generations.

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

```

##
## Kruskal-Wallis rank sum test
##
## data: uni_str_pos by K
## Kruskal-Wallis chi-squared = 260.74, df = 5, p-value < 2.2e-16

```

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

```

pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$K, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: coverage$uni_str_pos and coverage$K
##
##   1     2     4     8     15
## 2 4.4e-16 -     -     -
## 4 < 2e-16 < 2e-16 -     -     -
## 8 < 2e-16 4.5e-16 1.00 -     -
## 15 < 2e-16 6.4e-15 0.38 1.00 -
## 30 7.2e-10 1.0e-12 < 2e-16 < 2e-16 < 2e-16
##
## P value adjustment method: bonferroni

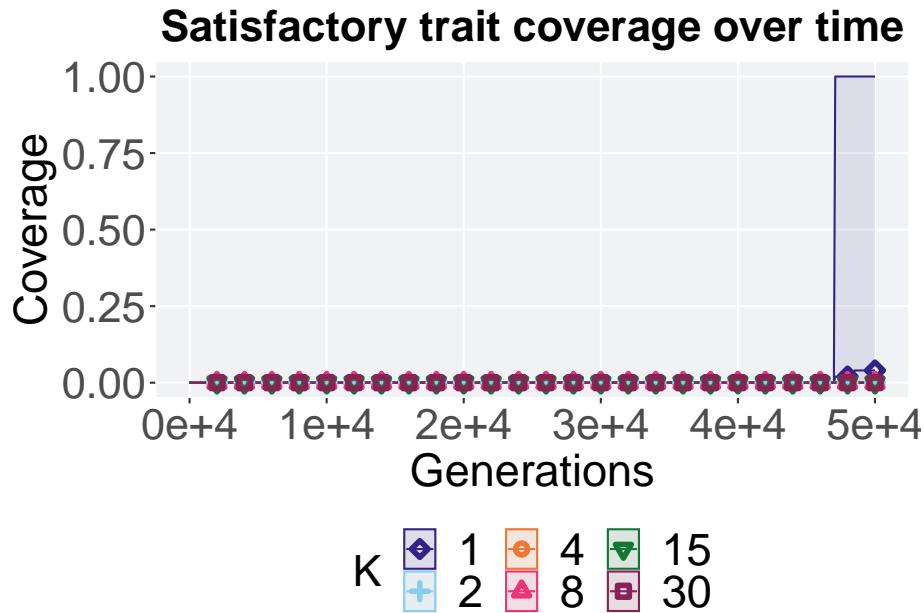
```

11.3.3 Multi-valley crossing

```
lines = filter(nov_ot_mvc, diagnostic == 'contradictory_objectives') %>%
  group_by(K, gen) %>%
  dplyr::summarise(
    min = min(pop_uni_obj),
    mean = mean(pop_uni_obj),
    max = max(pop_uni_obj)
  )
```

11.3.3.1 Satisfactory trait coverage over time

```
## `summarise()` has grouped output by 'K'. You can override using the `groups`  
## argument.  
  
ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +  
  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"  
    ) +  
    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
```



11.3.3.2 Satisfactory trait coverage comparison

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

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

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

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

  geom_point(data = end, aes(x = K, y=pop_uni_obj), col = mvc_col[2], position = position_nudge(x = .15), size = 1) +
  geom_boxplot(data = end, aes(x = K, y=pop_uni_obj), position = position_nudge(x = .15), lwd = 0.7, col = mvc_col[2])

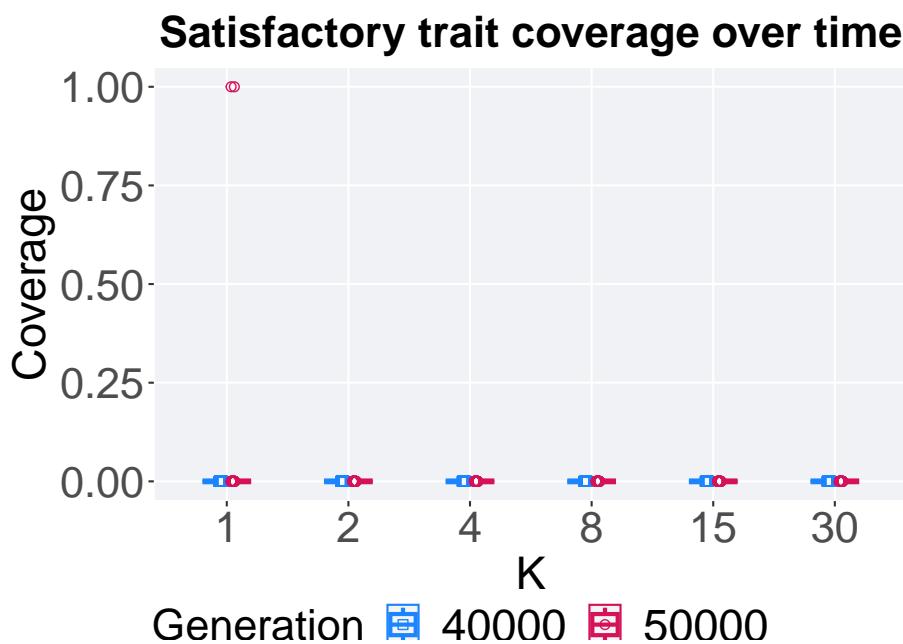
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="K"
  )+
  scale_shape_manual(values=c(0,1))+
```

```

scale_colour_manual(values = c(mvc_col[1],mvc_col[2])) +
  p_theme

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

```



11.3.3.2.1 Stats

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

```

slices = filter(nov_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(K, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(pop_uni_obj)),
    mean = mean(pop_uni_obj),
    min = min(pop_uni_obj),
    max = max(pop_uni_obj),
    median = median(pop_uni_obj),
    sd = sd(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 'K'. You can override using the `groups` argument.

## # A tibble: 12 x 9
## # Groups: K [6]
##   K     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>      <int> <int> <dbl> <dbl> <int> <dbl>
## 1 1     50000       50     0     0     0  0.04     1     0
## 2 1     40000       50     0     0     0     0     0     0
## 3 2     50000       50     0     0     0     0     0     0
## 4 2     40000       50     0     0     0     0     0     0
## 5 4     50000       50     0     0     0     0     0     0
## 6 4     40000       50     0     0     0     0     0     0
## 7 8     50000       50     0     0     0     0     0     0
## 8 8     40000       50     0     0     0     0     0     0
## 9 15    50000       50     0     0     0     0     0     0
## 10 15   40000       50     0     0     0     0     0     0
## 11 30   50000       50     0     0     0     0     0     0
## 12 30   40000       50     0     0     0     0     0     0

K 1

wilcox.test(x = filter(slices, K == 1 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, K == 1 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 1 & Generation == 50000)$pop_uni_obj and filter(slices,
## W = 1300, p-value = 0.1594
## alternative hypothesis: true location shift is not equal to 0

K 2

wilcox.test(x = filter(slices, K == 2 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, K == 2 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

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

```

```

## data: filter(slices, K == 2 & Generation == 50000)$pop_uni_obj and filter(slices, K == 2 & Ge
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
K 4
wilcox.test(x = filter(slices, K == 4 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, K == 4 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 4 & Generation == 50000)$pop_uni_obj and filter(slices, K == 4 & Ge
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
K 8
wilcox.test(x = filter(slices, K == 8 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, K == 8 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 8 & Generation == 50000)$pop_uni_obj and filter(slices, K == 8 & Ge
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
K 15
wilcox.test(x = filter(slices, K == 15 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, K == 15 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 15 & Generation == 50000)$pop_uni_obj and filter(slices, K == 15 &
## W = 1250, p-value = NA
## alternative hypothesis: true location shift is not equal to 0
K 30
wilcox.test(x = filter(slices, K == 30 & Generation == 50000)$pop_uni_obj,
             y = filter(slices, K == 30 & Generation == 40000)$pop_uni_obj,
             alternative = 't')

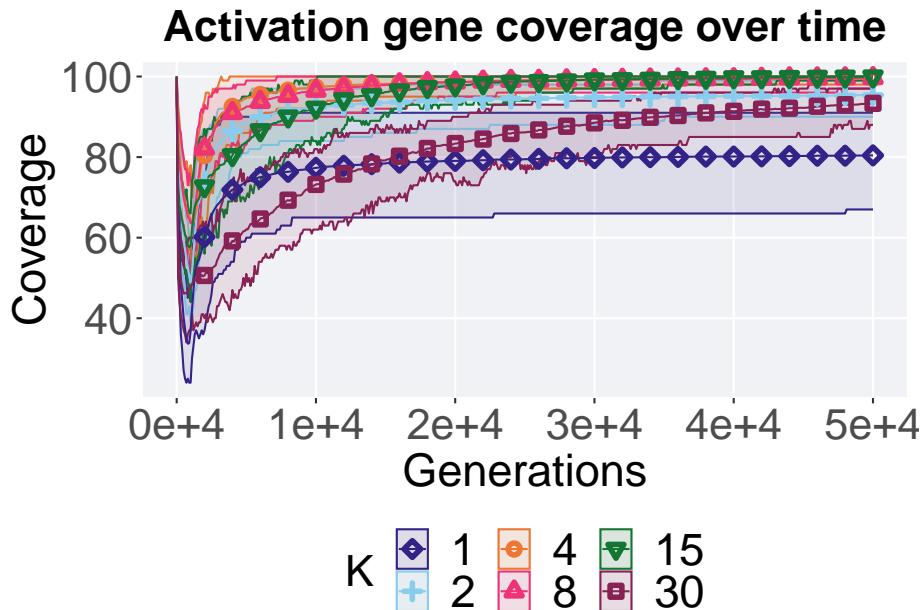
##
## Wilcoxon rank sum test with continuity correction

```

```
##  
## data: filter(slices, K == 30 & Generation == 50000)$pop_uni_obj and filter(slices,  
## W = 1250, p-value = NA  
## alternative hypothesis: true location shift is not equal to 0
```

```
lines = filter(nov_ot_mvc, diagnostic == 'contradictory_objectives') %>%  
  group_by(K, gen) %>%  
  dplyr::summarise(  
    min = min(uni_str_pos),  
    mean = mean(uni_str_pos),  
    max = max(uni_str_pos)  
  )
```

```
## `summarise()` has grouped output by 'K'. You can override using the `groups`  
## argument.  
ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +  
  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"  
  ) +  
  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
```



11.3.3.4 Activation gene coverage comparison

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

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

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

mvc_p = ggplot(mid, aes(x = K, y=uni_str_pos, group = K, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 1) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

  geom_point(data = end, aes(x = K, y=uni_str_pos), col = mvc_col[2], position = position_jitternudge(jitter.width = .03, nudge.x = 0.15), size = 1) +
  geom_boxplot(data = end, aes(x = K, y=uni_str_pos), position = position_nudge(x = .15, y = 0), lwd = 0.7, col = mvc_col[2], fill = mvc_col[2])

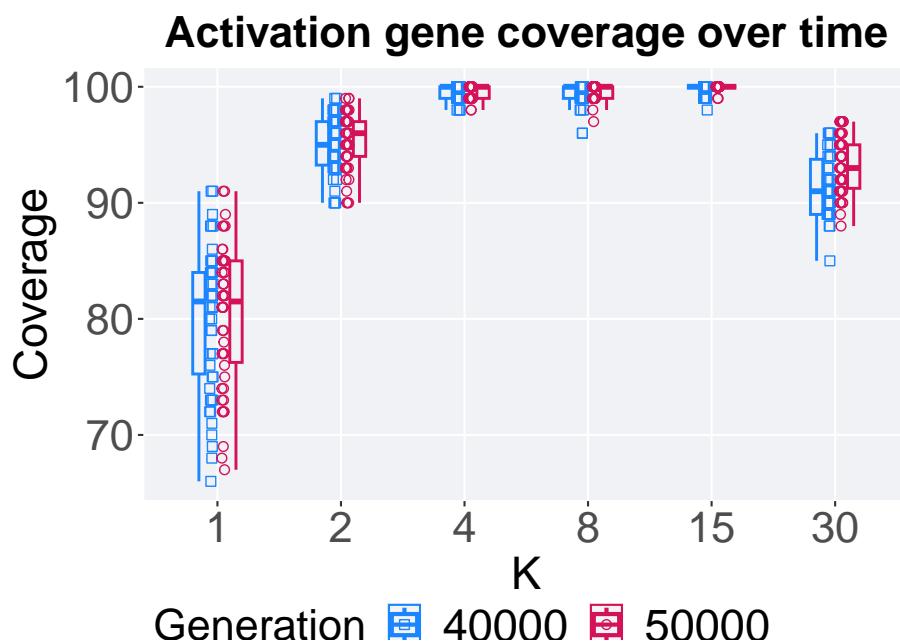
  scale_y_continuous(
    name="Coverage",
  ) +
  scale_x_discrete(
    name="K"
  )+
  scale_shape_manual(values=c(0,1))+
```

```

scale_colour_manual(values = c(mvc_col[1], mvc_col[2])) +
p_theme

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

```



11.3.3.4.1 Stats

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

```

slices = filter(nov_ot_mvc, diagnostic == 'contradictory_objectives' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(K, Generation) %>%
  dplyr::summarise(
    count = n(),
    na_cnt = sum(is.na(uni_str_pos)),
    mean_coverage = mean(univ_pos),
    median_coverage = median(univ_pos),
    min_coverage = min(univ_pos),
    max_coverage = max(univ_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 'K'. You can override using the `groups`  

## argument.

## # A tibble: 12 x 9
## # Groups: K [6]
##   K     Generation count na_cnt   min median   mean   max   IQR
##   <fct> <fct>      <int> <int> <dbl> <dbl> <int> <dbl>
## 1 1     50000       50    0    67   81.5  80.4    91   8.75
## 2 1     40000       50    0    66   81.5  80.1    91   8.75
## 3 2     50000       50    0    90   96    95.4    99   3
## 4 2     40000       50    0    90   95    95.1    99   3.75
## 5 4     50000       50    0    98   100   99.5   100   1
## 6 4     40000       50    0    98   100   99.4   100   1
## 7 8     50000       50    0    97   100   99.7   100   1
## 8 8     40000       50    0    96   100   99.4   100   1
## 9 15    50000       50    0    99   100   100.    100   0
## 10 15   40000       50    0    98   100   99.8   100   0
## 11 30    50000       50    0    88   93    93.3    97   3.75
## 12 30   40000       50    0    85   91    91.4    96   4.75

K 1
wilcox.test(x = filter(slices, K == 1 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 1 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##  

## Wilcoxon rank sum test with continuity correction
##  

## data: filter(slices, K == 1 & Generation == 50000)$uni_str_pos and filter(slices, K == 1 & Ge
## W = 1287, p-value = 0.8008
## alternative hypothesis: true location shift is not equal to 0

K 2
wilcox.test(x = filter(slices, K == 2 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 2 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##  

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

```

## data: filter(slices, K == 2 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1324.5, p-value = 0.6063
## alternative hypothesis: true location shift is not equal to 0

K 4
wilcox.test(x = filter(slices, K == 4 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 4 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 4 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1320, p-value = 0.5871
## alternative hypothesis: true location shift is not equal to 0

K 8
wilcox.test(x = filter(slices, K == 8 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 8 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 8 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1439.5, p-value = 0.1203
## alternative hypothesis: true location shift is not equal to 0

K 15
wilcox.test(x = filter(slices, K == 15 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 15 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 15 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1476, p-value = 0.007657
## alternative hypothesis: true location shift is not equal to 0

K 30
wilcox.test(x = filter(slices, K == 30 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 30 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction

```

```
##  
## data: filter(slices, K == 30 & Generation == 50000)$uni_str_pos and filter(slices, K == 30 &  
## W = 1785, p-value = 0.0002091  
## alternative hypothesis: true location shift is not equal to 0
```

11.4 Multi-path exploration results

Here we present the results for **best performances** and **activation gene coverage** found by each novelty search K value replicate on the multi-path exploration diagnostic. Best performance found refers to the largest average trait score found in a given population, while activation gene coverage refers to the count of unique activation genes in the population. Note that both values fall between 0 and 100.

11.4.1 Performance

Here we analyze the performances for each parameter replicate on the multi-path exploration diagnostic.

11.4.1.1 Performance over time

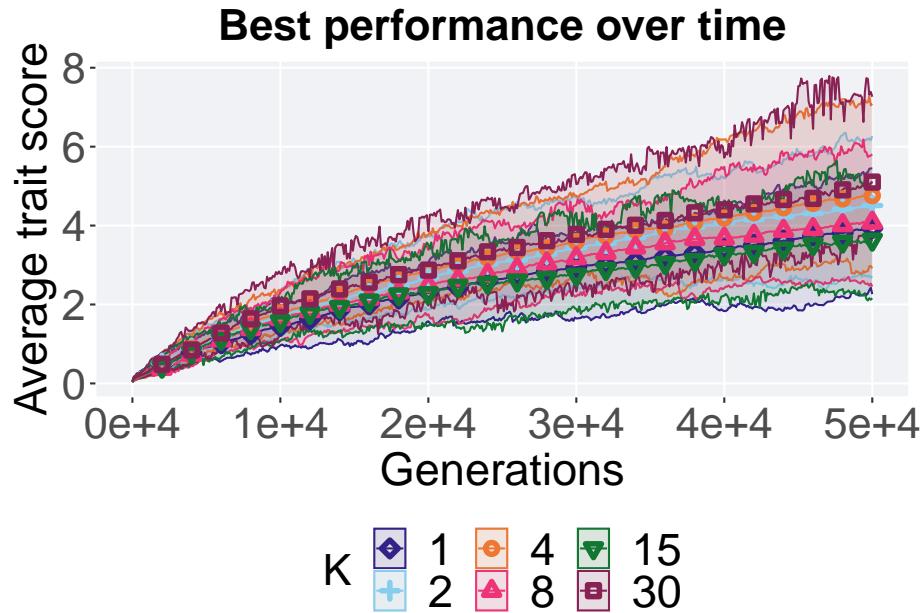
Performance over time.

```
lines = filter(nov_ot, diagnostic == 'multipath_exploration') %>%  
  group_by(K, gen) %>%  
  dplyr::summarise(  
    min = min(pop_fit_max),  
    mean = mean(pop_fit_max),  
    max = max(pop_fit_max)  
  )  
  
## `summarise()` has grouped output by 'K'. You can override using the `.groups`  
## argument.  
  
ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = K, fill = K, color = K, shape = K)) +  
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), 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="Average trait score"  
    ) +  
    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("Best performance over time") +
p_theme

```



11.4.1.2 Best performance throughout

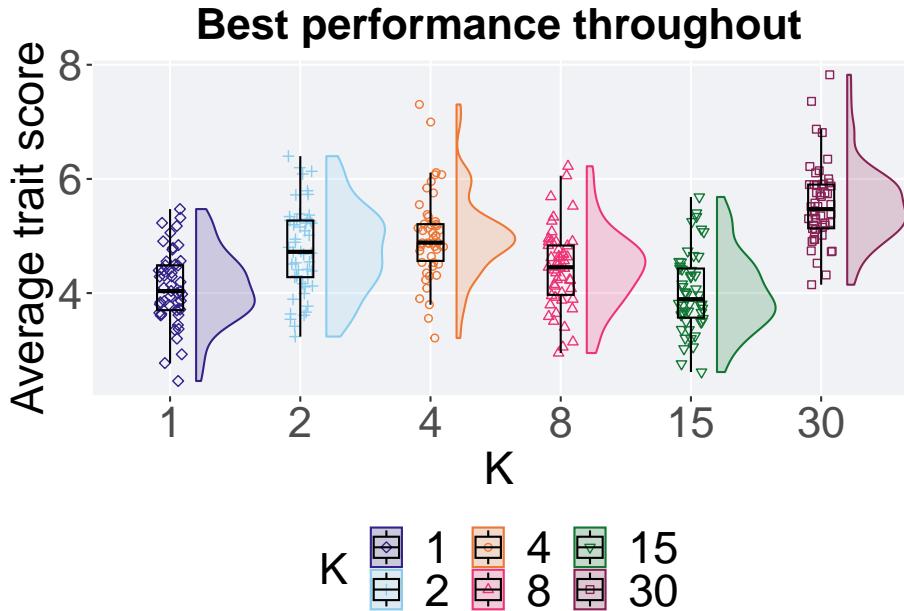
Here we plot the performance of the best performing solution found throughout 50,000 generations.

```

filter(nov_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration') %>%
ggplot(., aes(x = K, y = val / DIMENSIONALITY, color = K, fill = K, shape = K)) +
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5) +
  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"
  ) +
  scale_x_discrete(
    name = "K"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +

```

```
scale_fill_manual(values = cb_palette) +
  ggtitle("Best performance throughout") +
  p_theme
```



11.4.1.2.1 Stats

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

```
performance = filter(nov_best, col == 'pop_fit_max' & diagnostic == 'multipath_exploration')
group_by(performance, K) %>%
  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: 6 x 8
##   K      count na_cnt  min median  mean   max    IQR
##   <fct> <int>   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1       50      0  2.46  4.03  4.10  5.47  0.786
## 2 2       50      0  3.24  4.72  4.76  6.40  0.992
```

```
## 3 4      50      0  3.21   4.88   4.99   7.30 0.645
## 4 8      50      0  2.95   4.45   4.43   6.22 0.864
## 5 15     50      0  2.62   3.89   4.01   5.68 0.860
## 6 30     50      0  4.15   5.47   5.55   7.82 0.765
```

Kruskal–Wallis test provides evidence of statistical differences among the best performing solution found throughout 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: val by K
## Kruskal-Wallis chi-squared = 112.24, df = 5, p-value < 2.2e-16
```

Results for post-hoc Wilcoxon rank-sum test with a Bonferroni correction on the best performing solution found throughout 50,000 generations.

```
pairwise.wilcox.test(x = performance$val, g = performance$K, p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')
```

```
##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$val and performance$K
##
##    1      2      4      8      15
## 2  0.00070 -      -      -      -
## 4  3.9e-07 1.00000 -      -      -
## 8  0.21797 0.45226 0.00187 -      -
## 15 1.00000 0.00013 1.3e-07 0.03587 -
## 30 2.5e-13 2.8e-05 0.00061 4.2e-10 3.4e-13
##
## P value adjustment method: bonferroni
```

11.4.1.3 End of 50,000 generations

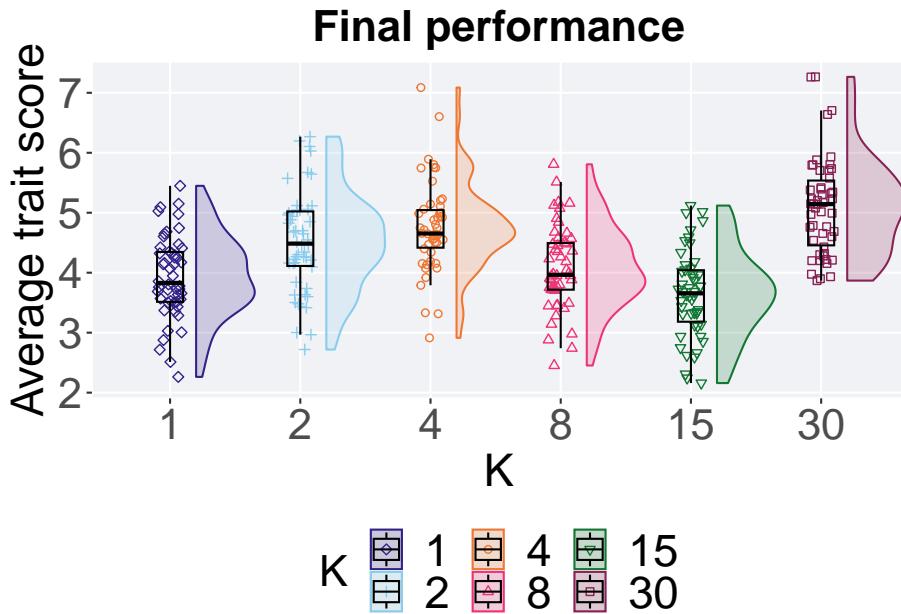
Best performance in the population at the end of 50,000 generations.

```
filter(nov_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = K, y = pop_fit_max / DIMENSIONALITY, color = K, fill = K, shape = K))
  geom_flat_violin(position = position_nudge(x = .2, y = 0), scale = 'width', alpha = 0.5)
  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"
  ) +
  scale_x_discrete()
```

```

  name = "K"
) +
scale_shape_manual(values = SHAPE) +
scale_colour_manual(values = cb_palette) +
scale_fill_manual(values = cb_palette) +
ggtitle("Final performance") +
p_theme

```



11.4.1.3.1 Stats

Summary statistics for the best performance in the population at the end of 50,000 generations.

```

performance = filter(nov_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(performance, K) %>%
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: 6 x 8

```

```
##   K      count na_cnt    min median   mean    max    IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1      50     0  2.26  3.83  3.93  5.45 0.832
## 2 2      50     0  2.72  4.48  4.51  6.27 0.910
## 3 4      50     0  2.91  4.65  4.76  7.09 0.627
## 4 8      50     0  2.45  3.96  4.08  5.81 0.777
## 5 15     50     0  2.16  3.66  3.64  5.12 0.859
## 6 30     50     0  3.87  5.14  5.10  7.27 1.08
```

Kruskal–Wallis test provides evidence of statistical differences among best performance in the population at the end of 50,000 generations.

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

```
##
## Kruskal-Wallis rank sum test
##
## data: pop_fit_max by K
## Kruskal-Wallis chi-squared = 95.737, df = 5, p-value < 2.2e-16
```

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

```
pairwise.wilcox.test(x = performance$pop_fit_max, g = performance$K , p.adjust.method =
  paired = FALSE, conf.int = FALSE, alternative = 't')

##
## Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data: performance$pop_fit_max and performance$K
##
##   1      2      4      8      15
## 2 0.01055 -      -      -      -
## 4 4.1e-06 1.00000 -      -      -
## 8 1.00000 0.16610 0.00013 -      -
## 15 0.63500 1.5e-05 5.2e-09 0.05030 -
## 30 2.3e-09 0.01195 0.38606 1.5e-07 4.0e-12
##
## P value adjustment method: bonferroni
```

11.4.2 Activation gene coverage

Here we analyze the activation gene coverage for each parameter replicate on the multi-path exploration diagnostic.

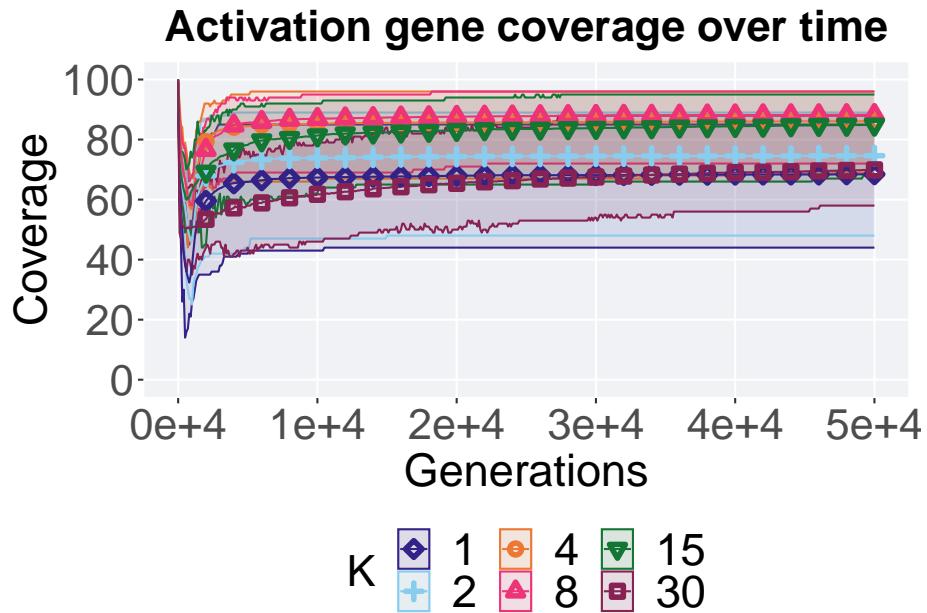
11.4.2.1 Coverage over time

Activation gene coverage over time.

```
lines = filter(nov_ot, diagnostic == 'multipath_exploration') %>%
  group_by(K, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),
    max = max(uni_str_pos)
  )

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

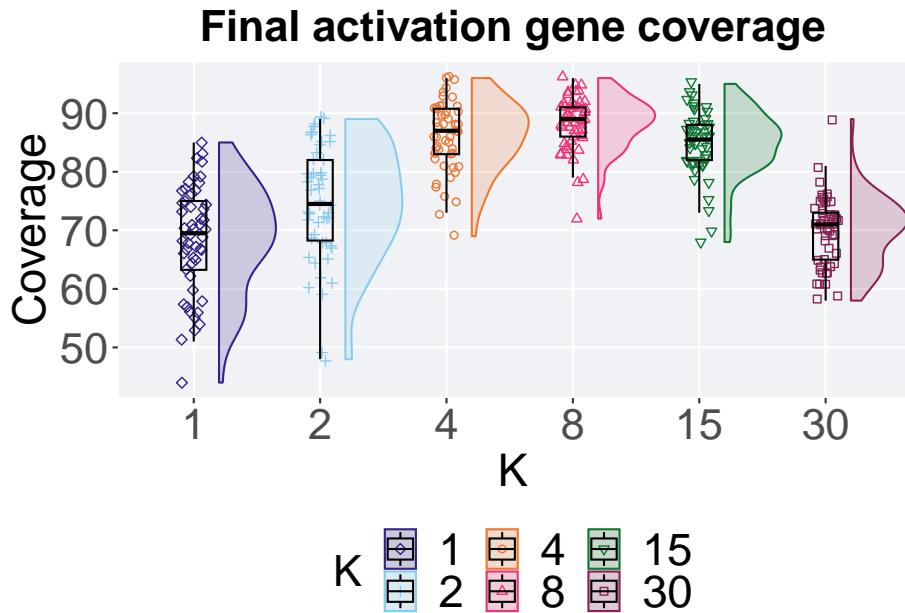
ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +
  geom_ribbon(aes(ymin = min, ymax = max), alpha = 0.1) +
  geom_line(size = 0.5) +
  geom_point(data = filter(lines, gen %% 2000 == 0 & gen != 0), size = 1.5, stroke = 2.0, alpha =
  scale_y_continuous(
    name="Coverage",
    limits=c(-1, 101),
    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
```



11.4.2.2 End of 50,000 generations

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

```
filter(nov_ot, diagnostic == 'multipath_exploration' & gen == 50000) %>%
  ggplot(., aes(x = K, y = uni_str_pos, color = K, fill = K, shape = K)) +
  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"
  ) +
  scale_x_discrete(
    name="K"
  ) +
  scale_shape_manual(values=SHAPE) +
  scale_colour_manual(values = cb_palette) +
  scale_fill_manual(values = cb_palette) +
  ggtitle("Final activation gene coverage") +
  p_theme
```



11.4.2.2.1 Stats

Summary statistics for the activation gene coverage in the population at the end of 50,000 generations.

```
coverage = filter(nov_ot, diagnostic == 'multipath_exploration' & gen == 50000)
group_by(coverage, K) %>%
  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: 6 x 8
##   K      count na_cnt  min median  mean  max   IQR
##   <fct> <int> <int> <dbl> <dbl> <dbl> <int> <dbl>
## 1 1        50     0    44    69.5  68.4   85  11.8
## 2 2        50     0    48    74.5  74.6   89  13.8
## 3 4        50     0    69    87    86.1   96  7.75
## 4 8        50     0    72    89    88.2   96   5
## 5 15       50     0    68    85.5  84.9   95   6
## 6 30       50     0    58    71    69.9   89   8
```

Kruskal–Wallis test provides evidence of statistical differences among activation gene coverage in the population at the end of 50,000 generations.

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

##
##  Kruskal-Wallis rank sum test
##
## data:  uni_str_pos by K
## Kruskal-Wallis chi-squared = 176.13, df = 5, p-value < 2.2e-16
```

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

```
pairwise.wilcox.test(x = coverage$uni_str_pos, g = coverage$K , p.adjust.method = "bonferroni",
                      paired = FALSE, conf.int = FALSE, alternative = 't')

##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  coverage$uni_str_pos and coverage$K
##
##    1      2      4      8      15
## 2  0.025   -     -     -     -
## 4  9.0e-14 5.3e-08  -     -     -
## 8  1.3e-15 2.4e-11 0.880   -     -
## 15 2.7e-13 8.4e-07 1.000  0.017   -
## 30 1.000   0.035   2.5e-14 1.5e-15 1.3e-13
##
## P value adjustment method: bonferroni
```

11.4.3 Multi-valley crossing

11.4.3.1 Performance over time

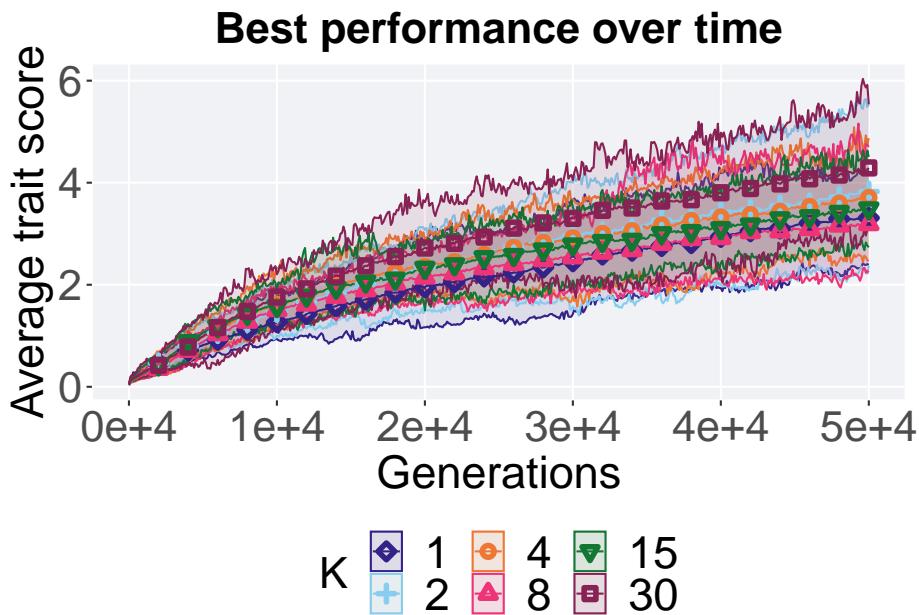
```
lines = filter(nov_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(K, gen) %>%
  dplyr::summarise(
    min = min(pop_fit_max),
    mean = mean(pop_fit_max),
    max = max(pop_fit_max)
  )

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

ggplot(lines, aes(x=gen, y=mean / DIMENSIONALITY, group = K, fill = K, shape = K,
  geom_ribbon(aes(ymin = min / DIMENSIONALITY, ymax = max / DIMENSIONALITY), alpha = 0.2))
```

```

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="Average trait score"
    ) +
    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("Best performance over time") +
    p_theme
  
```



11.4.3.2 Performance comparisons

```

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

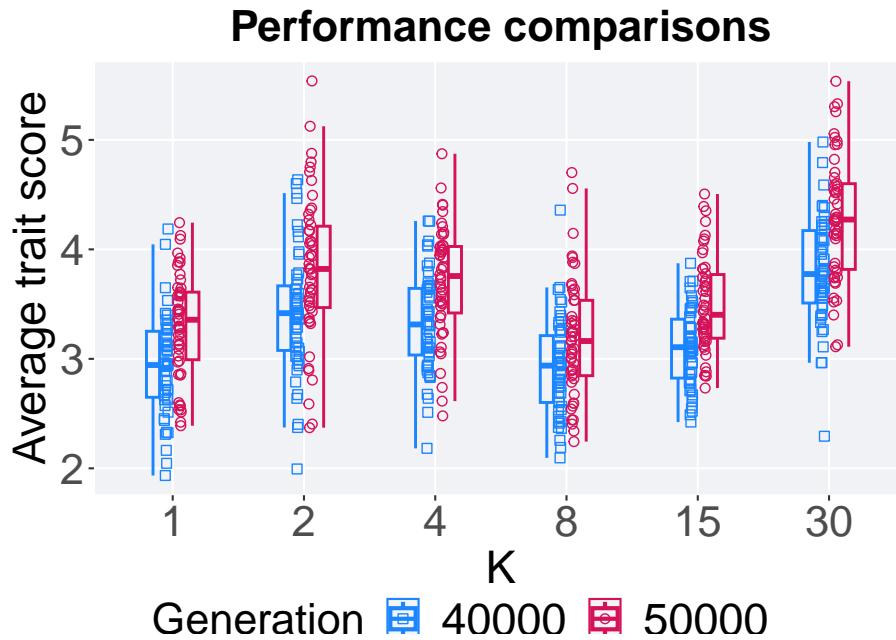
```
mid = filter(nov_ot_mvc, diagnostic == 'multipath_exploration' & gen == 40000)
mid$Generation <- factor(mid$gen)

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

  geom_point(data = end, aes(x = K, y=pop_fit_max / DIMENSIONALITY), col = mvc_col[2])
  geom_boxplot(data = end, aes(x = K, y=pop_fit_max / DIMENSIONALITY), position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[2])

  scale_y_continuous(
    name="Average trait score"
  ) +
  scale_x_discrete(
    name="K"
  ) +
  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
)
```



11.4.3.2.1 Stats

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

```
# 80% and final generation comparison
slices = filter(nov_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(K, 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 'K'. You can override using the `.groups` argument.

## # A tibble: 12 x 9
## # Groups:   K [6]
##   K     Generation count  na_cnt   min  median   mean   max    IQR
##   <dbl>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
```

```

##   <fct> <fct>     <int>  <int> <dbl>  <dbl> <dbl> <dbl>
## 1 1    50000      50     0  2.39  3.36  3.31  4.24 0.618
## 2 1    40000      50     0  1.93  2.94  2.94  4.19 0.603
## 3 2    50000      50     0  2.37  3.82  3.83  5.54 0.743
## 4 2    40000      50     0  1.99  3.42  3.42  4.64 0.590
## 5 4    50000      50     0  2.48  3.76  3.71  4.87 0.607
## 6 4    40000      50     0  2.18  3.31  3.33  4.26 0.609
## 7 8    50000      50     0  2.24  3.16  3.19  4.70 0.688
## 8 8    40000      50     0  2.10  2.94  2.92  4.36 0.611
## 9 15   50000      50     0  2.73  3.40  3.49  4.51 0.582
## 10 15  40000      50     0  2.42  3.11  3.10  3.87 0.538
## 11 30   50000      50     0  3.11  4.27  4.29  5.54 0.783
## 12 30   40000      50     0  2.29  3.77  3.80  4.98 0.662

K 1
wilcox.test(x = filter(slices, K == 1 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 1 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 1 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1808, p-value = 0.0001214
## alternative hypothesis: true location shift is not equal to 0

K 2
wilcox.test(x = filter(slices, K == 2 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 2 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 2 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1726, p-value = 0.001045
## alternative hypothesis: true location shift is not equal to 0

K 4
wilcox.test(x = filter(slices, K == 4 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 4 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 4 & Generation == 50000)$pop_fit_max and filter(slices,
## W = 1726, p-value = 0.001045
## alternative hypothesis: true location shift is not equal to 0

```

```

## W = 1788, p-value = 0.000211
## alternative hypothesis: true location shift is not equal to 0

K 8
wilcox.test(x = filter(slices, K == 8 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 8 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 8 & Generation == 50000)$pop_fit_max and filter(slices, K == 8 & Ge
## W = 1619, p-value = 0.01107
## alternative hypothesis: true location shift is not equal to 0

K 15
wilcox.test(x = filter(slices, K == 15 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 15 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 15 & Generation == 50000)$pop_fit_max and filter(slices, K == 15 &
## W = 1872, p-value = 1.831e-05
## alternative hypothesis: true location shift is not equal to 0

K 30
wilcox.test(x = filter(slices, K == 30 & Generation == 50000)$pop_fit_max,
             y = filter(slices, K == 30 & Generation == 40000)$pop_fit_max,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 30 & Generation == 50000)$pop_fit_max and filter(slices, K == 30 &
## W = 1859, p-value = 2.73e-05
## alternative hypothesis: true location shift is not equal to 0

```

11.4.3.3 Activation gene coverage over time

```

lines = filter(nov_ot_mvc, diagnostic == 'multipath_exploration') %>%
  group_by(K, gen) %>%
  dplyr::summarise(
    min = min(uni_str_pos),
    mean = mean(uni_str_pos),

```

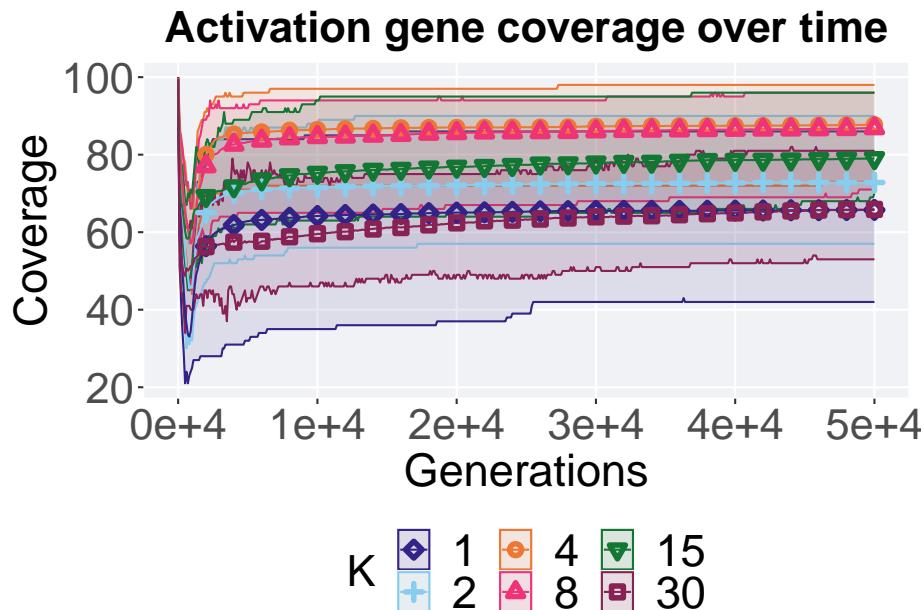
```

    max = max(uni_str_pos)
)

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

ggplot(lines, aes(x=gen, y=mean, group = K, fill = K, color = K, shape = K)) +
  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"
  ) +
  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

```



11.4.3.4 Activation gene coverage comparison

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

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

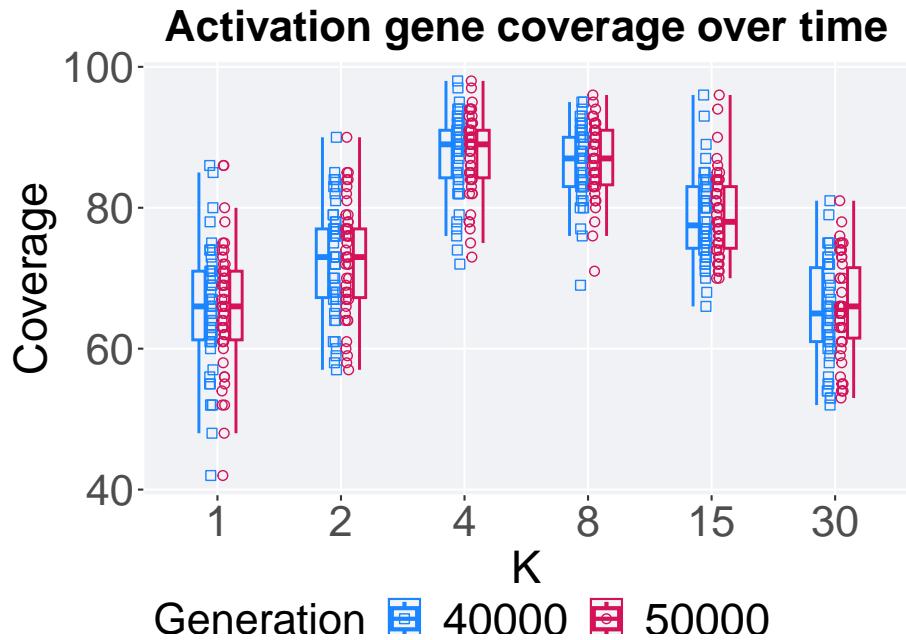
mid = filter(nov_ot_mvc, diagnostic == 'multipath_exploration' & gen == 40000)
mid$Generation <- factor(mid$gen)

mvc_p = ggplot(mid, aes(x = K, y=uni_str_pos, group = K, shape = Generation)) +
  geom_point(col = mvc_col[1] , position = position_jitternudge(jitter.width = .03, nudge.x = -0.15), size = 100) +
  geom_boxplot(position = position_nudge(x = -.15, y = 0), lwd = 0.7, col = mvc_col[1], fill = mvc_col[1])

geom_point(data = end, aes(x = K, y=uni_str_pos), col = mvc_col[2], position = position_jitternudge(x = 0.15, y = 0)) +
  geom_boxplot(data = end, aes(x = K, y=uni_str_pos), position = position_nudge(x = .15, y = 0), col = mvc_col[2], fill = mvc_col[2])

scale_y_continuous(
  name="Coverage",
) +
  scale_x_discrete(
    name="K"
) +
  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 over time") +
    theme(legend.position="none"),
  legend,
  nrow=2,
  rel_heights = c(1,.05),
  label_size = TSIZE
)
```



11.4.3.4.1 Stats

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

```
slices = filter(nov_ot_mvc, diagnostic == 'multipath_exploration' & (gen == 50000 | gen == 40000))
slices$Generation <- factor(slices$gen, levels = c(50000,40000))
slices %>%
  group_by(K, 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 'K'. You can override using the `.groups` argument.

## # A tibble: 12 x 9
## # Groups:   K [6]
##       K     Generation count  na_cnt   min median   mean   max    IQR
##   <fct> <fct>     <int>   <int>   <int>   <dbl>   <dbl> <int>   <dbl>
```

```

## 1 1    50000      50     0    42    66   65.7    86   9.75
## 2 1    40000      50     0    42    66   65.6    86   9.75
## 3 2    50000      50     0    57    73   72.8    90   9.75
## 4 2    40000      50     0    57    73   72.7    90   9.75
## 5 4    50000      50     0    73    89   87.7    98   6.75
## 6 4    40000      50     0    72    89   87.5    98   6.75
## 7 8    50000      50     0    71    87   86.9    96   7.75
## 8 8    40000      50     0    69    87   86.5    95   7
## 9 15   50000      50     0    70    78   79.1    96   8.75
## 10 15  40000      50     0    66    77.5  78.6    96   8.75
## 11 30   50000      50     0    53    66   65.9    81   10
## 12 30   40000      50     0    52    65   65.0    81   10.5

K 1
wilcox.test(x = filter(slices, K == 1 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 1 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 1 & Generation == 50000)$uni_str_pos and filter(slices, K == 1 & Ge
## W = 1262, p-value = 0.9367
## alternative hypothesis: true location shift is not equal to 0

K 2
wilcox.test(x = filter(slices, K == 2 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 2 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 2 & Generation == 50000)$uni_str_pos and filter(slices, K == 2 & Ge
## W = 1257.5, p-value = 0.9614
## alternative hypothesis: true location shift is not equal to 0

K 4
wilcox.test(x = filter(slices, K == 4 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 4 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 4 & Generation == 50000)$uni_str_pos and filter(slices, K == 4 & Ge
## W = 1287, p-value = 0.8007

```

```

## alternative hypothesis: true location shift is not equal to 0
K 8
wilcox.test(x = filter(slices, K == 8 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 8 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 8 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1304, p-value = 0.7115
## alternative hypothesis: true location shift is not equal to 0

K 15
wilcox.test(x = filter(slices, K == 15 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 15 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 15 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1295.5, p-value = 0.756
## alternative hypothesis: true location shift is not equal to 0

K 30
wilcox.test(x = filter(slices, K == 30 & Generation == 50000)$uni_str_pos,
             y = filter(slices, K == 30 & Generation == 40000)$uni_str_pos,
             alternative = 't')

##
## Wilcoxon rank sum test with continuity correction
##
## data: filter(slices, K == 30 & Generation == 50000)$uni_str_pos and filter(slices,
## W = 1363.5, p-value = 0.4346
## alternative hypothesis: true location shift is not equal to 0

```