Supplemental Material for Environmental connectivity influences long-term evolutionary outcomes

2025-08-11

Contents

Chapter 1

Introduction

This is the supplemental material our 2025 Artificial Life Conference paper, "Environmental connectivity influences long-term evolutionary outcomes". This is not intended as a stand-alone document, but as a companion to our main manuscript.

1.1 About our supplemental material

Our supplemental material is hosted using GitHub pages. We compiled our data analyses and supplemental documentation into this web-accessible book using bookdown.

The source code and configuration files for this supplemental material can be found in this GitHub repository.

Our supplemental material includes the following:

- Data availability (Section 2)
- Local compilation instructions (Section 3)
- TODO

1.2 Contributing authors

- Grant Gordon
- Austin J. Ferguson
- Emily Dolson
- Alexander Lalejini

Chapter 2

Data availability

2.1 Source code

The source code for his work is publicly accessible on GitHub: https://github.com/amlalejini/alife-2025-env-conn. This repository has also been archived on Zenodo: https://doi.org/10.5281/zenodo.16795777

2.2 Experiment results

Data generated from our experiments used in analyses are available online, archived in an OSF repository: https://osf.io/ahs6m/

On OSF, the following compressed archives contain the data presented in our manuscript:

- `2025-04-17-squished-lattice-longer-avida.tar.gz`
- `2025-04-17-vary-structs-avida.tar.gz`
- `squished-lattice-mabe.tar.gz`
- `vary-structs-mabe.tar.gz`

Chapter 3

Compilation instructions

Instructions for compiling and running the software used in this study on your local machine. All experiments were run on Mac or Linux-based operating systems.

You will need a C++ compiler that supports at least C++17. We used g++13 for all local compilations.

You will also need Python to run graph generation and analysis. Python dependencies are listed in the `requirements.txt` at the root of this repository.

Statistical analyses and data visualizations were conducted using R.

Experiments in our simplified model used the MABE2 software, and experiments with digital organisms (self-replicating computer programs) used a modified version of the Avida software platform.

3.1 Instructions

First, clone the `alife-2025-env-conn` repository (https://github.com/amlalejini/alife-2025-env-conn.git) to your machine. Then, initialize and update git submodule inside the repository. From inside the repository on your machine, run:

```
git submodule update --init --recursive
```

This will download and update the following dependencies:

- `avida-empirical` (commit hash: `266f95f8fcb452655330dab55caa9f1408b49ffa`): A modified implementation of the Avida software that supports the capacity to configure environmental connectivity.

- `evo_spatial_discoveries` (commit hash: `2c384e93df231125bae83fc6c38d8dc8c64eb6ee`): Contains configurations for MABE2 experiments.
- `MABE2` (commit hash: `4f8eb86f997ee89f6d0e0b1144c5be162f4d8d1b`): MABE = "Modular agent-based evolver", which is a software platform deigned to empower developers to easily build and customize software for evolutionary computation or artificial life. We used this platform to implement our non-avida experiments.
- `network_correlation` (commit hash: `9d9a07f7436c3569d10eb3b03c6b30e1238c74ef`): Third-party python implementations of various graph statistics and analyses.

To compile Avida, navigate into the `third-party/avida-empirical/` directory and run `./build_avida/`. The compiled executable will be created in the `third-party/avida-empirical/cbuild/work/` directory.

To compile MABE2, navigate into the `third-party/MABE2/build` directory and run `make native`. The compiled executable will be created in the `third-party/MABE2/build` directory.

Configuration files used Avida experiments can be found in the `experiments/` directory (within the `hpc/config` subdirectory for any given experiment). Configuration files used for MABE2 experiments can be found in `third-party/evo_spatial_discoveries/experiments/`.

Chapter 4

Simple model - Varied spatial structure experiment analyses

4.1 Dependencies and setup

```
library(tidyverse)
library(cowplot)
library(RColorBrewer)
library(khroma)
library(rstatix)
library(knitr)
library(kableExtra)
library(infer)
source("https://gist.githubusercontent.com/benmarwick/2a1bb0133ff568cbe28d/raw/fb53bd97121f7f9ce9
```

```
# Check if Rmd is being compiled using bookdown
bookdown <- exists("bookdown_build")
```

```
experiment_slug <- "vg-experiments"
working_directory <- paste(
  "experiments",
  "mabe2-exps",
  experiment_slug,
  sep = "/"
)
```

```
# Adjust working directory if being knitted for bookdown build.
if (bookdown) {
  working_directory <- paste0(
    bookdown_wd_prefix,
    working_directory
  )
}
```

```
# Configure our default graphing theme
theme_set(theme_cowplot())
# Create a directory to store plots
plot_dir <- paste(
  working_directory,
  "rmd_plots",
  sep = "/"
)

dir.create(
  plot_dir,
  showWarnings = FALSE
)
```

4.2 Max organism data analyses

```
max_generation <- 100000
max_org_data_path <- paste(
  working_directory,
  "data",
  "combined_max_org_data.csv",
  sep = "/"
)
# Data file has time series
max_org_data_ts <- read_csv(max_org_data_path)
max_org_data_ts <- max_org_data_ts %>%
  mutate(
    landscape = as.factor(landscape),
    structure = as.factor(structure),
  ) %>%
  mutate(
    valleys_crossed = case_when(
      landscape == "Valley crossing" ~ round(log(fitness, base = 1.5)),
      .default = 0
    )
  )
# Get tibble with just final generation
max_org_data <- max_org_data_ts %>%
  filter(generation == max_generation)
```

Check that replicate count for each condition matches expectations.

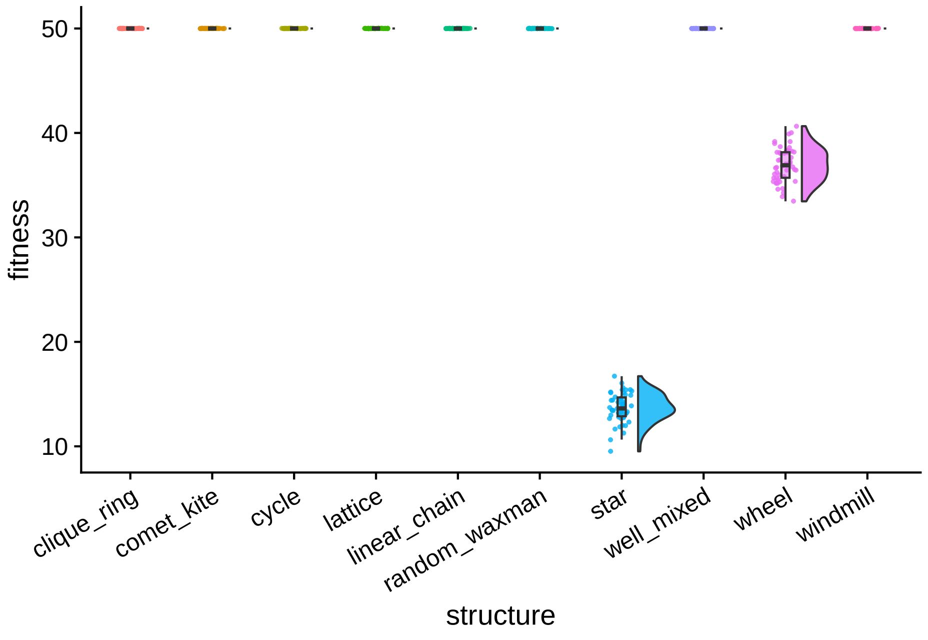
```
run_summary <- max_org_data %>%
  group_by(landscape, structure) %>%
  summarize(
    n = n()
  )
print(run_summary, n = 30)
```

```
## # A tibble: 30 x 3
## # Groups:   landscape [3]
##    landscape       structure         n
##    <fct>           <fct>         <int>
##  1 Multipath       clique_ring      50
##  2 Multipath       comet_kite       50
##  3 Multipath       cycle            50
##  4 Multipath       lattice          50
##  5 Multipath       linear_chain     50
##  6 Multipath       random_waxman    50
##  7 Multipath       star             50
##  8 Multipath       well_mixed       50
##  9 Multipath       wheel            50
## 10 Multipath       windmill         50
## 11 Single gradient clique_ring      50
## 12 Single gradient comet_kite       50
## 13 Single gradient cycle            50
## 14 Single gradient lattice          50
## 15 Single gradient linear_chain     50
## 16 Single gradient random_waxman    50
## 17 Single gradient star             50
## 18 Single gradient well_mixed       50
## 19 Single gradient wheel            50
## 20 Single gradient windmill         50
## 21 Valley crossing clique_ring      50
## 22 Valley crossing comet_kite       50
## 23 Valley crossing cycle            50
## 24 Valley crossing lattice          50
## 25 Valley crossing linear_chain     50
## 26 Valley crossing random_waxman    50
## 27 Valley crossing star             50
## 28 Valley crossing well_mixed       50
## 29 Valley crossing wheel            50
## 30 Valley crossing windmill         50
```

4.2.1 Fitness in smooth gradient landscape

Maximum fitness

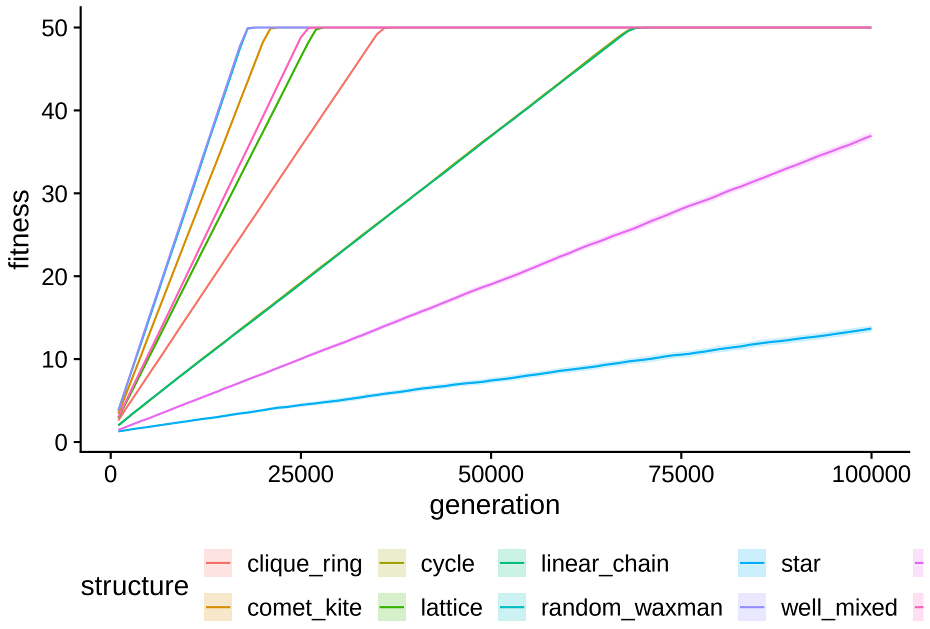
```
single_gradient_final_fitness_plt <- ggplot(
    data = filter(max_org_data, landscape == "Single gradient"),
    mapping = aes(
      x = structure,
      y = fitness,
      fill = structure
    )
  ) +
  geom_flat_violin(
    position = position_nudge(x = .2, y = 0),
    alpha = .8
  ) +
  geom_point(
    mapping = aes(color = structure),
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .1,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/single_gradient_final_fitness.pdf"),
  plot = single_gradient_final_fitness_plt,
  width = 15,
  height = 10
)

single_gradient_final_fitness_plt
```



Maximum fitness over time

```
single_gradient_fitness_ts_plt <- ggplot(
    data = filter(max_org_data_ts, landscape == "Single gradient"),
    mapping = aes(
      x = generation,
      y = fitness,
      color = structure,
      fill = structure
    )
  ) +
  stat_summary(fun = "mean", geom = "line") +
  stat_summary(
    fun.data = "mean_cl_boot",
    fun.args = list(conf.int = 0.95),
    geom = "ribbon",
    alpha = 0.2,
    linetype = 0
  ) +
  theme(legend.position = "bottom")
```

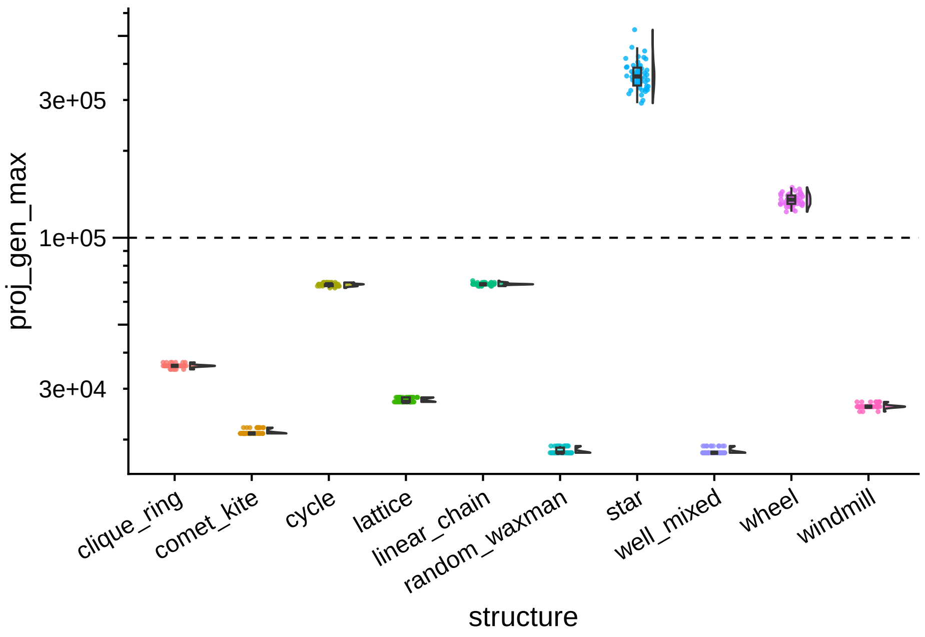
```
ggsave(
  plot = single_gradient_fitness_ts_plt,
  filename = paste0(
    plot_dir,
    "/single_gradient_fitness_ts.pdf"
  ),
  width = 15,
  height = 10
)

single_gradient_fitness_ts_plt
```



Time to maximum fitness

```
# Find all rows with maximum fitness value, then get row with minimum generation,
#  then project out expected generation to max (for runs that didn't finish)
max_possible_fit = 50
time_to_max_single_gradient <- max_org_data_ts %>%
  filter(landscape == "Single gradient") %>%
  group_by(rep, structure) %>%
  slice_max(
    fitness,
    n = 1
  ) %>%
  slice_min(
    generation,
    n = 1
  ) %>%
  mutate(
    proj_gen_max = (max_possible_fit / fitness) * generation
  )
```

```
single_gradient_gen_max_proj_plt <- ggplot(
    data = time_to_max_single_gradient,
    mapping = aes(
      x = structure,
      y = proj_gen_max,
      fill = structure
    )
  ) +
  geom_flat_violin(
    position = position_nudge(x = .2, y = 0),
    alpha = .8
  ) +
  geom_point(
    mapping = aes(color = structure),
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .1,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  scale_y_log10(
    guide = "axis_logticks"
  ) +
  # scale_y_continuous(
  #   trans="pseudo_log",
  #   breaks = c(10, 100, 1000, 10000, 100000, 1000000)
  #   ,limits = c(10, 100, 1000, 10000, 100000, 1000000)
  # ) +
  geom_hline(
    yintercept = max_generation,
    linetype = "dashed"
  ) +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/single_gradient_gen_max_proj.pdf"),
  plot = single_gradient_gen_max_proj_plt,
  width = 15,
  height = 10
)

single_gradient_gen_max_proj_plt
```



Rank ordering of time to max fitness values

```
time_to_max_single_gradient %>%
  group_by(structure) %>%
  summarize(
    reps = n(),
    median_proj_gen = median(proj_gen_max),
    mean_proj_gen = mean(proj_gen_max)
  ) %>%
  arrange(
    mean_proj_gen
  )
```

```
## # A tibble: 10 x 4
##    structure      reps median_proj_gen mean_proj_gen
##    <fct>         <int>           <dbl>         <dbl>
##  1 well_mixed       50           18000         18240
##  2 random_waxman    50           18000         18260
##  3 comet_kite       50           21000         21220
##  4 windmill         50           26000         26100
##  5 lattice          50           27000         27460
##  6 clique_ring      50           36000         36020
##  7 cycle            50           69000         68840
##  8 linear_chain     50           69000         69080
##  9 wheel            50          135481.       135502.
## 10 star             50          361785.       366603.
```

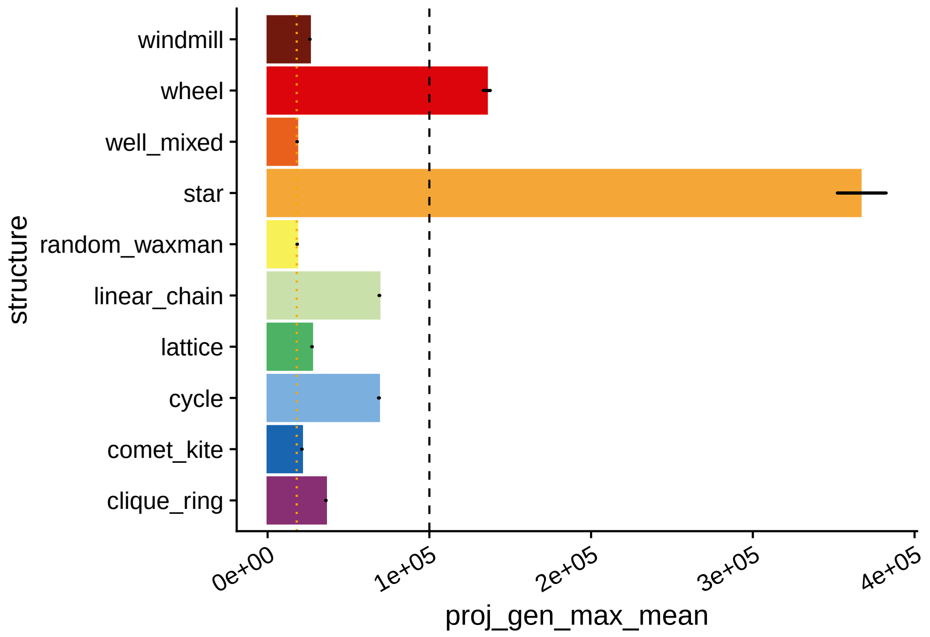
```
kruskal.test(
  formula = proj_gen_max ~ structure,
  data = time_to_max_single_gradient
)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  proj_gen_max by structure
## Kruskal-Wallis chi-squared = 490.93, df = 9, p-value < 2.2e-16
```

```
wc_results <- pairwise.wilcox.test(
  x = time_to_max_single_gradient$proj_gen_max,
  g = time_to_max_single_gradient$structure,
  p.adjust.method   = "holm",
  exact = FALSE
)

single_gradient_proj_gen_max_wc_table <- kbl(wc_results$p.value) %>%
  kable_styling()

save_kable(
  single_gradient_proj_gen_max_wc_table,
  paste0(plot_dir, "/single_gradient_proj_gen_max_wc_table.pdf")
)

single_gradient_proj_gen_max_wc_table
```

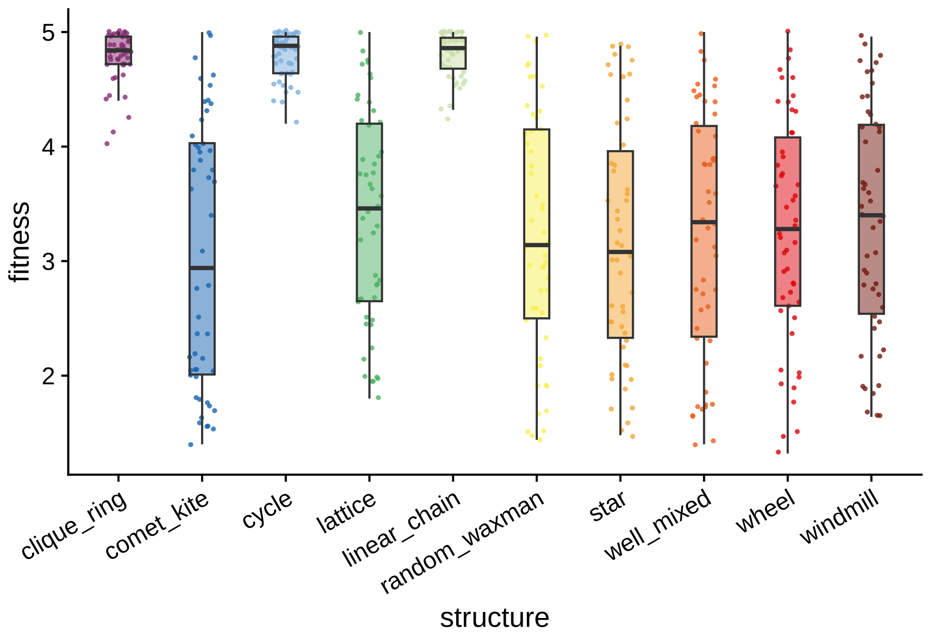
	clique_ring	comet_kite	cycle	lattice	linear_chain	random_waxma
comet_kite	0	NA	NA	NA	NA	N
cycle	0	0	NA	NA	NA	N
lattice	0	0	0.0000000	NA	NA	N
linear_chain	0	0	0.2915242	0	NA	N
random_waxman	0	0	0.0000000	0	0	N
star	0	0	0.0000000	0	0	0.000000
well_mixed	0	0	0.0000000	0	0	0.821833
wheel	0	0	0.0000000	0	0	0.000000
windmill	0	0	0.0000000	0	0	0.000000

```r
library(boot)
# Define sample mean function
samplemean <- function(x, d) {
  return(mean(x[d]))
}

summary_gen_to_max <- tibble(
  structure = character(),
  proj_gen_max_mean = double(),
  proj_gen_max_mean_ci_low = double(),
  proj_gen_max_mean_ci_high = double()
)

structures <- levels(time_to_max_single_gradient$structure)
for (struct in structures) {
  boot_result <- boot(
    data = filter(
      time_to_max_single_gradient,
      structure == struct
    )$proj_gen_max,
    statistic = samplemean,
    R = 10000
  )
  result_ci <- boot.ci(boot_result, conf = 0.99, type = "perc")
  m <- result_ci$t0
  low <- result_ci$percent[4]
  high <- result_ci$percent[5]

  summary_gen_to_max <- summary_gen_to_max %>%
    add_row(
      structure = struct,
      proj_gen_max_mean = m,
      proj_gen_max_mean_ci_low = low,
      proj_gen_max_mean_ci_high = high
    )
}

wm_median <- median(
  filter(time_to_max_single_gradient, structure == "well_mixed")$proj_gen_max
)

simple_time_to_max_plt <- ggplot(
    data = summary_gen_to_max,
    mapping = aes(
      x = structure,
      y = proj_gen_max_mean,
      fill = structure,
      color = structure
    )
  ) +
  # geom_point() +
  geom_col() +
  geom_linerange(
    aes(
      ymin = proj_gen_max_mean_ci_low,
      ymax = proj_gen_max_mean_ci_high
    ),
    color = "black",
    linewidth = 0.75,
    lineend = "round"
  ) +
  # scale_y_log10(
  #   guide = "axis_logticks"
  # ) +
  geom_hline(
    yintercept = max_generation,
    linetype = "dashed"
  ) +
  geom_hline(
    yintercept = wm_median,
    linetype = "dotted",
    color = "orange"
  ) +
  scale_color_discreterainbow() +
  scale_fill_discreterainbow() +
  coord_flip() +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/simple_time_to_max.pdf"),
  plot = simple_time_to_max_plt,
  width = 8,
  height = 4
)

simple_time_to_max_plt
```



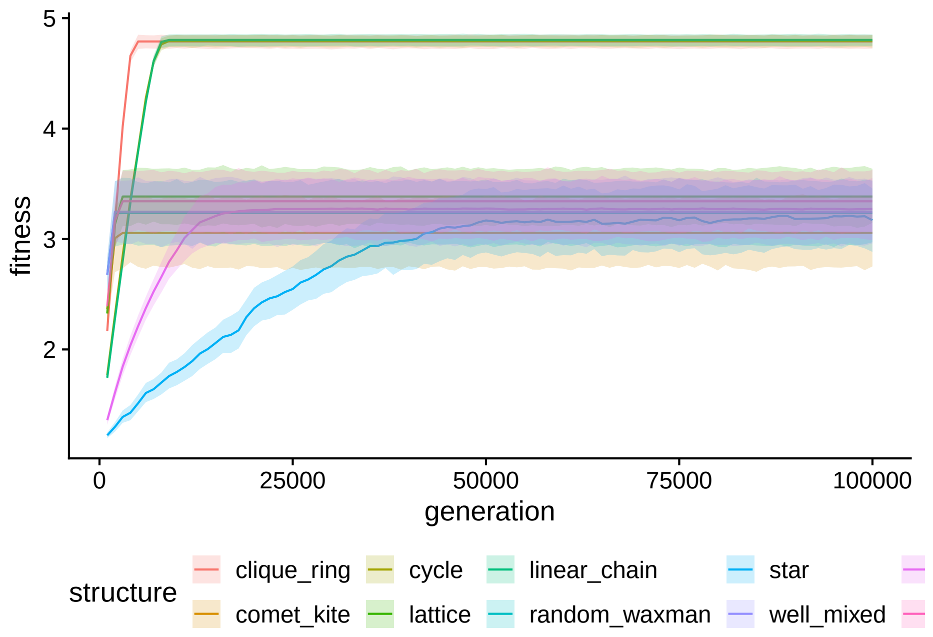
4.2.2 Fitness in multi-path landscape

```
multipath_final_fitness_plt <- ggplot(
    data = filter(max_org_data, landscape == "Multipath"),
    mapping = aes(
      x = structure,
      y = fitness,
      fill = structure
    )
  ) +
  # geom_flat_violin(
  #   position = position_nudge(x = .2, y = 0),
  #   alpha = .8
  # ) +
  geom_point(
    mapping = aes(color = structure),
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .3,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  scale_color_discreterainbow() +
  scale_fill_discreterainbow() +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/multipath_final_fitness.pdf"),
  plot = multipath_final_fitness_plt,
  width = 6,
  height = 4
)

multipath_final_fitness_plt
```



Max fitness over time

```
multipath_fitness_ts_plt <- ggplot(
    data = filter(max_org_data_ts, landscape == "Multipath"),
    mapping = aes(
      x = generation,
      y = fitness,
      color = structure,
      fill = structure
    )
  ) +
  stat_summary(fun = "mean", geom = "line") +
  stat_summary(
    fun.data = "mean_cl_boot",
    fun.args = list(conf.int = 0.95),
    geom = "ribbon",
    alpha = 0.2,
    linetype = 0
  ) +
  theme(legend.position = "bottom")

ggsave(
  plot = multipath_fitness_ts_plt,
  filename = paste0(
    plot_dir,
    "/multipath_fitness_ts.pdf"
  ),
  width = 15,
  height = 10
)

multipath_fitness_ts_plt
```



Rank ordering of fitness values

```
max_org_data %>%
  filter(landscape == "Multipath") %>%
  group_by(structure) %>%
  summarize(
    reps = n(),
    median_fitness = median(fitness),
    mean_fitness = mean(fitness)
  ) %>%
  arrange(
    desc(mean_fitness)
  )
```

```
## # A tibble: 10 x 4
##    structure      reps median_fitness mean_fitness
##    <fct>         <int>          <dbl>        <dbl>
##  1 linear_chain     50           4.86         4.80
##  2 cycle            50           4.88         4.79
##  3 clique_ring      50           4.84         4.79
##  4 lattice          50           3.46         3.38
##  5 windmill         50           3.4          3.34
##  6 wheel            50           3.28         3.27
##  7 well_mixed       50           3.34         3.25
##  8 random_waxman    50           3.14         3.23
##  9 star             50           3.08         3.17
## 10 comet_kite       50           2.94         3.06
```

```
kruskal.test(
  formula = fitness ~ structure,
  data = filter(max_org_data, landscape == "Multipath")
)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  fitness by structure
## Kruskal-Wallis chi-squared = 246.11, df = 9, p-value < 2.2e-16
```

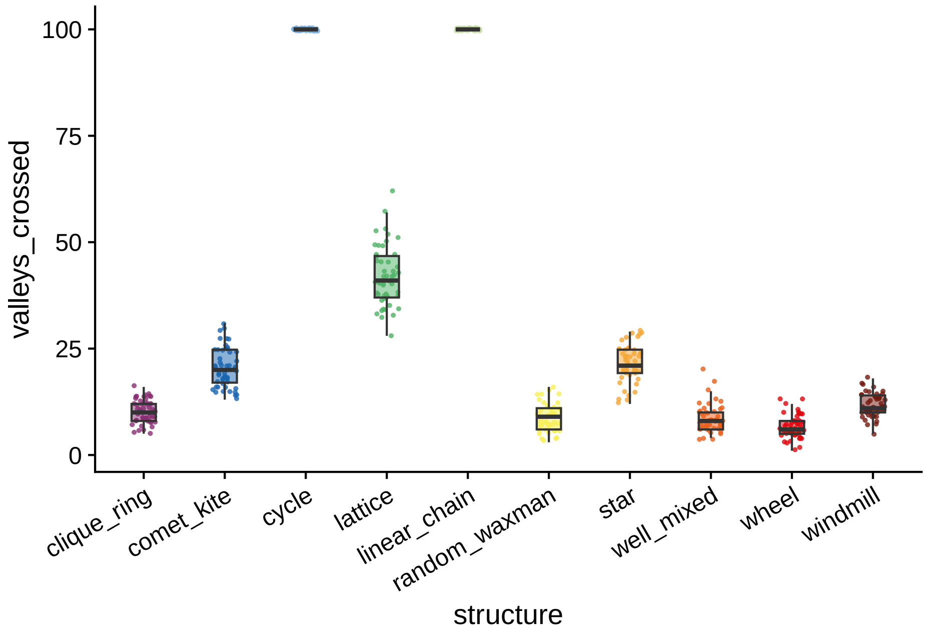
```
wc_results <- pairwise.wilcox.test(
  x = filter(max_org_data, landscape == "Multipath")$fitness,
  g = filter(max_org_data, landscape == "Multipath")$structure,
  p.adjust.method   = "holm",
  exact = FALSE
)

mp_fitness_wc_table <- kbl(wc_results$p.value) %>%
  kable_styling()

save_kable(
  mp_fitness_wc_table,
  paste0(plot_dir, "/multipath_fitness_wc_table.pdf")
)

mp_fitness_wc_table
```

4.2.3 Valleys crossed in valley-crossing landscape

	clique_ring	comet_kite	cycle	lattice	linear_chain	random_waxman	star	well_
comet_kite	0	NA	NA	NA	NA	NA	NA	
cycle	1	0	NA	NA	NA	NA	NA	
lattice	0	1	0	NA	NA	NA	NA	
linear_chain	1	0	1	0	NA	NA	NA	
random_waxman	0	1	0	1	0	NA	NA	
star	0	1	0	1	0	1	NA	
well_mixed	0	1	0	1	0	1	1	
wheel	0	1	0	1	0	1	1	
windmill	0	1	0	1	0	1	1	

```r
valleycrossing_valleys_plt <- ggplot(
    data = filter(max_org_data, landscape == "Valley crossing"),
    mapping = aes(
      x = structure,
      y = valleys_crossed,
      fill = structure
    )
  ) +
  # geom_flat_violin(
  #   position = position_nudge(x = .2, y = 0),
  #   alpha = .8
  # ) +
  geom_point(
    mapping = aes(color = structure),
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .3,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  scale_color_discreterainbow() +
  scale_fill_discreterainbow() +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )
```

```
ggsave(
  filename = paste0(plot_dir, "/valleycrossing_valleys_crossed.pdf"),
  plot = valleycrossing_valleys_plt,
  width = 6,
  height = 4
)

valleycrossing_valleys_plt
```



Rank ordering of fitness values

```
vc <- max_org_data %>%
  filter(landscape == "Valley crossing") %>%
  group_by(structure) %>%
  summarize(
    reps = n(),
    median_valleys_crossed = median(valleys_crossed),
    mean_valleys_crossed = mean(valleys_crossed),
    min_valleys_crossed = min(valleys_crossed)
  ) %>%
  arrange(
    desc(mean_valleys_crossed)
  )
vc
```

```
## # A tibble: 10 x 5
##    structure      reps median_valleys_crossed mean_valleys_crossed
##    <fct>         <int>                  <dbl>                <dbl>
##  1 cycle            50                    100               100
##  2 linear_chain     50                    100               100
##  3 lattice          50                     41                41.9
##  4 star             50                     21                21.5
##  5 comet_kite       50                     20                20.5
##  6 windmill         50                     11                11.6
##  7 clique_ring      50                     10                10.3
##  8 random_waxman    50                      9                 8.76
##  9 well_mixed       50                      8                 8.46
## 10 wheel            50                      6                 6.6
## # i 1 more variable: min_valleys_crossed <dbl>
```

```
vc$min_valleys_crossed
```

```
##  [1] 100 100  28  12  13   5   5   3   4   1
```

```
kruskal.test(
  formula = valleys_crossed ~ structure,
  data = filter(max_org_data, landscape == "Valley crossing")
)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  valleys_crossed by structure
## Kruskal-Wallis chi-squared = 444.04, df = 9, p-value < 2.2e-16
```

```
wc_results <- pairwise.wilcox.test(
  x = filter(max_org_data, landscape == "Valley crossing")$valleys_crossed,
  g = filter(max_org_data, landscape == "Valley crossing")$structure,
  p.adjust.method   = "holm",
  exact = FALSE
)

vc_valleys_crossed_wc_table <- kbl(wc_results$p.value) %>%
  kable_styling()

save_kable(
  vc_valleys_crossed_wc_table,
  paste0(plot_dir, "/valley_crossing_valleys_wc_table.pdf")
)
```

	clique_ring	comet_kite	cycle	lattice	linear_chain	random_waxman
comet_kite	0.0000000	NA	NA	NA	NA	NA
cycle	0.0000000	0.0000000	NA	NA	NA	NA
lattice	0.0000000	0.0000000	0	NA	NA	NA
linear_chain	0.0000000	0.0000000	NaN	0	NA	NA
random_waxman	0.0414336	0.0000000	0	0	0	NA
star	0.0000000	0.4016992	0	0	0	0.0000000
well_mixed	0.0028498	0.0000000	0	0	0	0.4620430
wheel	0.0000001	0.0000000	0	0	0	0.0029961
windmill	0.0895493	0.0000000	0	0	0	0.0001323

```
vc_valleys_crossed_wc_table
```

Chapter 5

Simple model - Squished toroid experiment analyses

5.1 Setup and Dependencies

```
library(tidyverse)
library(cowplot)
library(RColorBrewer)
library(khroma)
library(rstatix)
library(knitr)
library(kableExtra)
library(infer)
source("https://gist.githubusercontent.com/benmarwick/2a1bb0133ff568cbe28d/raw/fb53bd97121f7f9ce9
```

```
# Check if Rmd is being compiled using bookdown
bookdown <- exists("bookdown_build")
```

```
experiment_slug <- "lattice-experiments"
working_directory <- paste(
  "experiments",
  "mabe2-exps",
  experiment_slug,
  sep = "/"
)
# Adjust working directory if being knitted for bookdown build.
if (bookdown) {
  working_directory <- paste0(
    bookdown_wd_prefix,
    working_directory
  )
}
```

```
# Configure our default graphing theme
theme_set(theme_cowplot())
# Create a directory to store plots
plot_dir <- paste(
  working_directory,
  "rmd_plots",
  sep = "/"
)

dir.create(
  plot_dir,
  showWarnings = FALSE
)
```

5.2 Max organism data analyses

```
max_generation <- 100000
max_org_data_path <- paste(
  working_directory,
  "data",
  "combined_max_org_data.csv",
  sep = "/"
)
# Data file has time series
max_org_data_ts <- read_csv(max_org_data_path)
max_org_data_ts <- max_org_data_ts %>%
  mutate(
    landscape = as.factor(landscape),
    structure = factor(
      structure,
      levels = c(
        "1_3600",
        "2_1800",
        "3_1200",
        "4_900",
        "15_240",
        "30_120",
        "60_60"
      )
    ),
  ) %>%
  mutate(
    valleys_crossed = case_when(
      landscape == "Valley crossing" ~ round(log(fitness, base = 1.5)),
      .default = 0
    )
  )
# Get tibble with just final generation
max_org_data <- max_org_data_ts %>%
  filter(generation == max_generation)
```

Check that replicate count for each condition matches expectations.

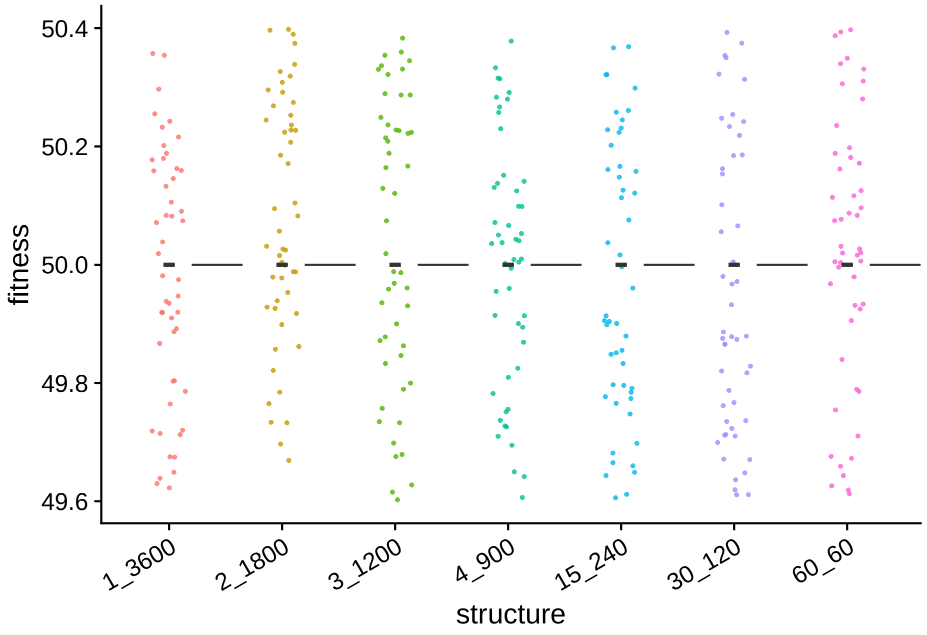
```
run_summary <- max_org_data %>%
  group_by(landscape, structure) %>%
  summarize(
    n = n()
  )
print(run_summary, n = 30)
```

```
## # A tibble: 21 x 3
## # Groups:   landscape [3]
##    landscape       structure     n
##    <fct>           <fct>     <int>
##  1 Multipath       1_3600       50
##  2 Multipath       2_1800       50
##  3 Multipath       3_1200       50
##  4 Multipath       4_900        50
##  5 Multipath       15_240       50
##  6 Multipath       30_120       50
##  7 Multipath       60_60        50
##  8 Single gradient 1_3600       50
##  9 Single gradient 2_1800       50
## 10 Single gradient 3_1200       50
## 11 Single gradient 4_900        50
## 12 Single gradient 15_240       50
## 13 Single gradient 30_120       50
## 14 Single gradient 60_60        50
## 15 Valley crossing 1_3600       50
## 16 Valley crossing 2_1800       50
## 17 Valley crossing 3_1200       50
## 18 Valley crossing 4_900        50
## 19 Valley crossing 15_240       50
## 20 Valley crossing 30_120       50
## 21 Valley crossing 60_60        50
```

5.2.1 Fitness in smooth gradient landscape

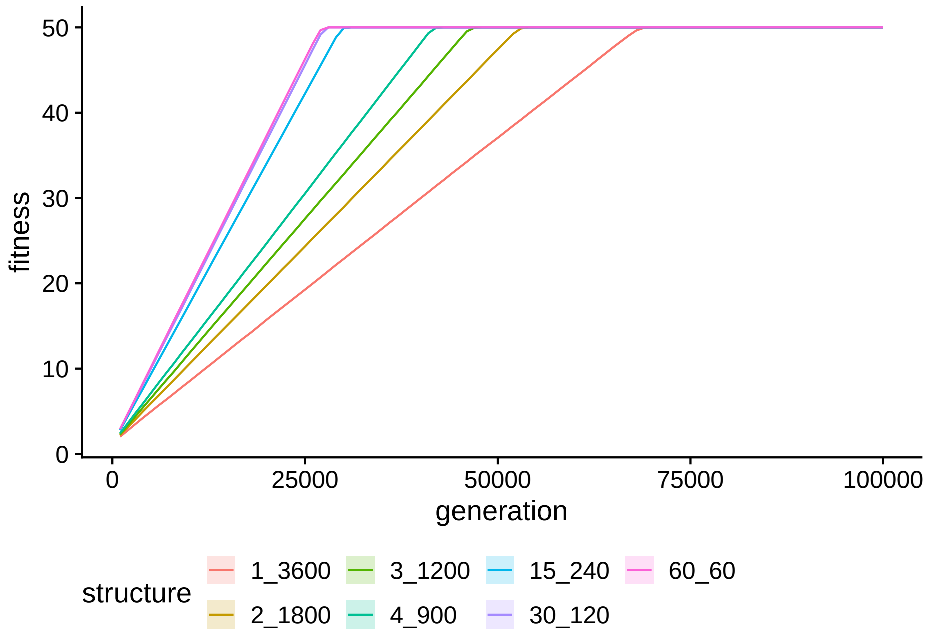
```
single_gradient_final_fitness_plt <- ggplot(
    data = filter(max_org_data, landscape == "Single gradient"),
    mapping = aes(
      x = structure,
      y = fitness,
      fill = structure
    )
  ) +
  geom_flat_violin(
    position = position_nudge(x = .2, y = 0),
    alpha = .8
  ) +
  geom_point(
    mapping = aes(color = structure),
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .1,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/single_gradient_final_fitness.pdf"),
  plot = single_gradient_final_fitness_plt,
  width = 15,
  height = 10
)
```

```
single_gradient_final_fitness_plt
```



Max fitness over time

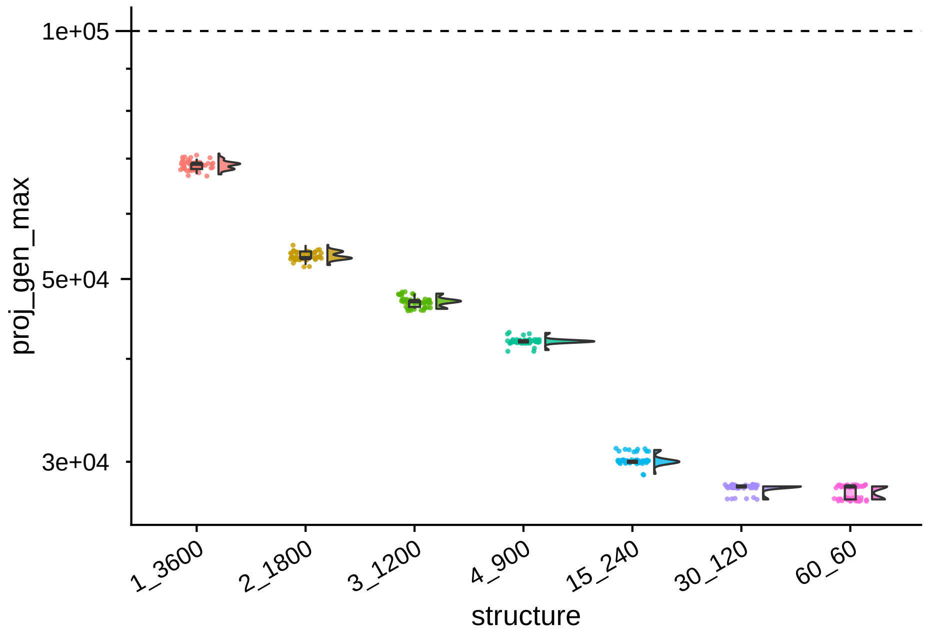
```
single_gradient_fitness_ts_plt <- ggplot(
    data = filter(max_org_data_ts, landscape == "Single gradient"),
    mapping = aes(
      x = generation,
      y = fitness,
      color = structure,
      fill = structure
    )
  ) +
  stat_summary(fun = "mean", geom = "line") +
  stat_summary(
    fun.data = "mean_cl_boot",
    fun.args = list(conf.int = 0.95),
    geom = "ribbon",
    alpha = 0.2,
    linetype = 0
  ) +
  theme(legend.position = "bottom")

ggsave(
  plot = single_gradient_fitness_ts_plt,
  filename = paste0(
    plot_dir,
    "/single_gradient_fitness_ts.pdf"
  ),
  width = 15,
  height = 10
)

single_gradient_fitness_ts_plt
```



Time to maximum fitness

```
# Find all rows with maximum fitness value, then get row with minimum generation,
#  then project out expected generation to max (for runs that didn't finish)
max_possible_fit = 50
time_to_max_single_gradient <- max_org_data_ts %>%
  filter(landscape == "Single gradient") %>%
  group_by(rep, structure) %>%
  slice_max(
    fitness,
    n = 1
  ) %>%
  slice_min(
    generation,
    n = 1
  ) %>%
  mutate(
    proj_gen_max = (max_possible_fit / fitness) * generation
  )
```

```
single_gradient_gen_max_proj_plt <- ggplot(
    data = time_to_max_single_gradient,
    mapping = aes(
      x = structure,
      y = proj_gen_max,
      fill = structure
    )
  ) +
  geom_flat_violin(
    position = position_nudge(x = .2, y = 0),
    alpha = .8
  ) +
  geom_point(
    mapping = aes(color = structure),
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .1,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  scale_y_log10(
    guide = "axis_logticks"
  ) +
  # scale_y_continuous(
  #   trans="pseudo_log",
  #   breaks = c(10, 100, 1000, 10000, 100000, 1000000)
  #   ,limits = c(10, 100, 1000, 10000, 100000, 1000000)
  # ) +
  geom_hline(
    yintercept = max_generation,
    linetype = "dashed"
  ) +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/single_gradient_gen_max_proj.pdf"),
  plot = single_gradient_gen_max_proj_plt,
  width = 15,
  height = 10
)

single_gradient_gen_max_proj_plt
```



Rank ordering of time to max fitness values

```
time_to_max_single_gradient %>%
  group_by(structure) %>%
  summarize(
    reps = n(),
    median_proj_gen = median(proj_gen_max),
    mean_proj_gen = mean(proj_gen_max)
  ) %>%
  arrange(
    mean_proj_gen
  )
```

```
## # A tibble: 7 x 4
##   structure  reps median_proj_gen mean_proj_gen
##   <fct>     <int>           <dbl>         <dbl>
## 1 60_60        50           28000         27540
## 2 30_120       50           28000         27880
## 3 15_240       50           30000         30160
## 4 4_900        50           42000         42020
## 5 3_1200       50           47000         46900
## 6 2_1800       50           53000         53340
## 7 1_3600       50           69000         68700
```

```
kruskal.test(
  formula = proj_gen_max ~ structure,
  data = time_to_max_single_gradient
)
```

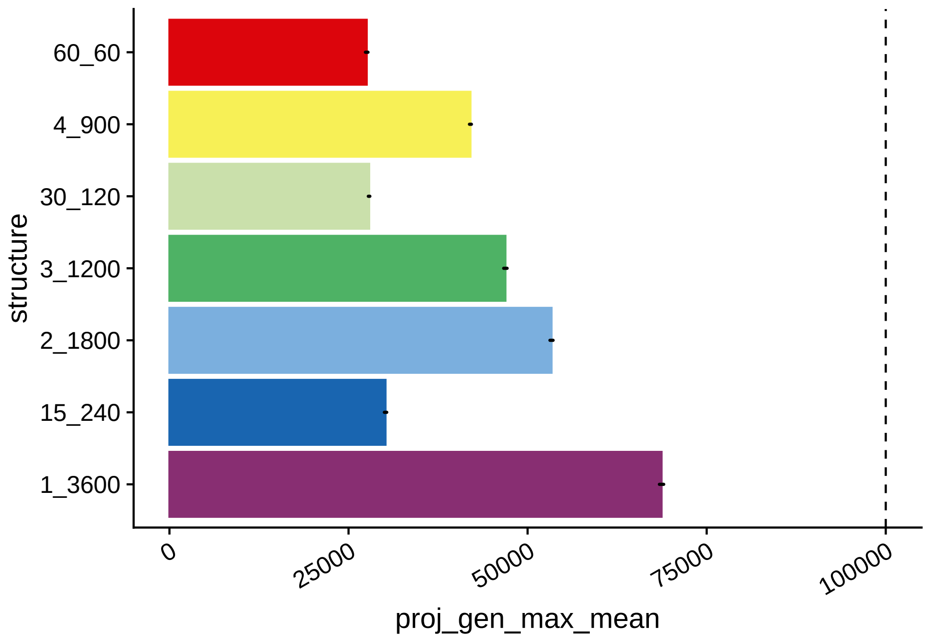
```
##
##  Kruskal-Wallis rank sum test
##
## data:  proj_gen_max by structure
## Kruskal-Wallis chi-squared = 341.17, df = 6, p-value < 2.2e-16
```

```
wc_results <- pairwise.wilcox.test(
  x = time_to_max_single_gradient$proj_gen_max,
  g = time_to_max_single_gradient$structure,
  p.adjust.method   = "holm",
  exact = FALSE
)

single_gradient_proj_gen_max_wc_table <- kbl(wc_results$p.value) %>%
  kable_styling()

save_kable(
  single_gradient_proj_gen_max_wc_table,
  paste0(plot_dir, "/single_gradient_proj_gen_max_wc_table.pdf")
)
single_gradient_proj_gen_max_wc_table
```

```
library(boot)
# Define sample mean function
samplemean <- function(x, d) {
  return(mean(x[d]))
}
```

	1_3600	2_1800	3_1200	4_900	15_240	30_120
2_1800	0	NA	NA	NA	NA	NA
3_1200	0	0	NA	NA	NA	NA
4_900	0	0	0	NA	NA	NA
15_240	0	0	0	0	NA	NA
30_120	0	0	0	0	0	NA
60_60	0	0	0	0	0	0.0001966

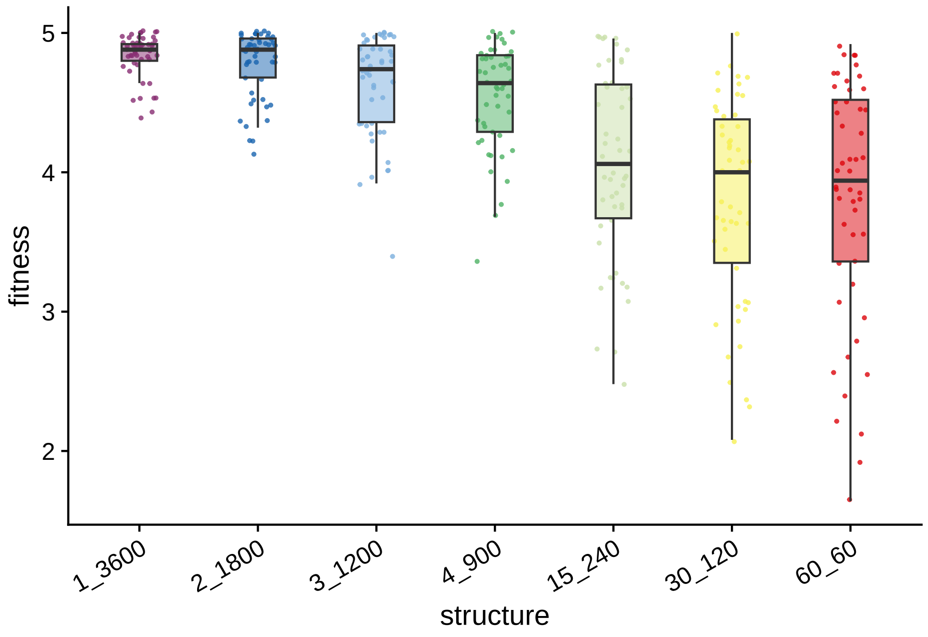
```r
summary_gen_to_max <- tibble(
  structure = character(),
  proj_gen_max_mean = double(),
  proj_gen_max_mean_ci_low = double(),
  proj_gen_max_mean_ci_high = double()
)

structures <- levels(time_to_max_single_gradient$structure)
for (struct in structures) {
  boot_result <- boot(
    data = filter(
      time_to_max_single_gradient,
      structure == struct
    )$proj_gen_max,
    statistic = samplemean,
    R = 10000
  )
  result_ci <- boot.ci(boot_result, conf = 0.99, type = "perc")
  m <- result_ci$t0
  low <- result_ci$percent[4]
  high <- result_ci$percent[5]

  summary_gen_to_max <- summary_gen_to_max %>%
    add_row(
      structure = struct,
      proj_gen_max_mean = m,
      proj_gen_max_mean_ci_low = low,
      proj_gen_max_mean_ci_high = high
    )
}

wm_median <- median(
  filter(time_to_max_single_gradient, structure == "well_mixed")$proj_gen_max
)

simple_time_to_max_plt <- ggplot(
    data = summary_gen_to_max,
    mapping = aes(
      x = structure,
      y = proj_gen_max_mean,
      fill = structure,
      color = structure
    )
  ) +
  # geom_point() +
  geom_col() +
  geom_linerange(
    aes(
      ymin = proj_gen_max_mean_ci_low,
      ymax = proj_gen_max_mean_ci_high
    ),
    color = "black",
    linewidth = 0.75,
    lineend = "round"
  ) +
  # scale_y_log10(
  #   guide = "axis_logticks"
  # ) +
  geom_hline(
    yintercept = max_generation,
    linetype = "dashed"
  ) +
  geom_hline(
    yintercept = wm_median,
    linetype = "dotted",
    color = "orange"
  ) +
  scale_color_discreterainbow() +
  scale_fill_discreterainbow() +
  coord_flip() +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/simple_time_to_max.pdf"),
  plot = simple_time_to_max_plt,
  width = 8,
  height = 4
)

simple_time_to_max_plt
```



5.2.2 Fitness in multi-path landscape

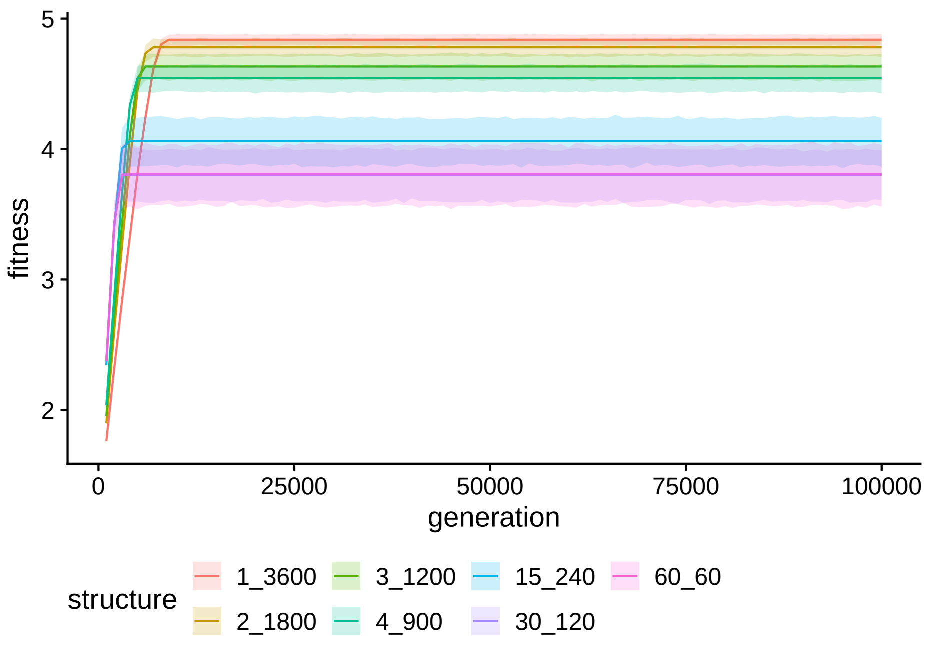
```
multipath_final_fitness_plt <- ggplot(
    data = filter(max_org_data, landscape == "Multipath"),
    mapping = aes(
      x = structure,
      y = fitness,
      fill = structure
    )
  ) +
  # geom_flat_violin(
  #   position = position_nudge(x = .2, y = 0),
  #   alpha = .8
  # ) +
  geom_point(
    mapping = aes(color = structure),
```

```r
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .3,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  scale_color_discreterainbow() +
  scale_fill_discreterainbow() +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )

ggsave(
  filename = paste0(plot_dir, "/multipath_final_fitness.pdf"),
  plot = multipath_final_fitness_plt,
  width = 6,
  height = 4
)

multipath_final_fitness_plt
```



Max fitness over time

```
multipath_fitness_ts_plt <- ggplot(
    data = filter(max_org_data_ts, landscape == "Multipath"),
    mapping = aes(
      x = generation,
      y = fitness,
      color = structure,
      fill = structure
    )
  ) +
  stat_summary(fun = "mean", geom = "line") +
  stat_summary(
    fun.data = "mean_cl_boot",
    fun.args = list(conf.int = 0.95),
    geom = "ribbon",
    alpha = 0.2,
    linetype = 0
  ) +
  theme(legend.position = "bottom")

ggsave(
  plot = multipath_fitness_ts_plt,
  filename = paste0(
    plot_dir,
    "/multipath_fitness_ts.pdf"
  ),
  width = 15,
  height = 10
)

multipath_fitness_ts_plt
```



Rank ordering of fitness values

```
max_org_data %>%
  filter(landscape == "Multipath") %>%
  group_by(structure) %>%
  summarize(
    reps = n(),
    median_fitness = median(fitness),
    mean_fitness = mean(fitness)
  ) %>%
  arrange(
    desc(mean_fitness)
  )
```

```
## # A tibble: 7 x 4
##   structure  reps median_fitness mean_fitness
##   <fct>     <int>          <dbl>         <dbl>
## 1 1_3600       50           4.88          4.84
## 2 2_1800       50           4.88          4.78
## 3 3_1200       50           4.74          4.63
## 4 4_900        50           4.64          4.54
## 5 15_240       50           4.06          4.06
## 6 60_60        50           3.94          3.81
## 7 30_120       50           4             3.80
```

```
kruskal.test(
  formula = fitness ~ structure,
  data = filter(max_org_data, landscape == "Multipath")
)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  fitness by structure
## Kruskal-Wallis chi-squared = 144.73, df = 6, p-value < 2.2e-16
```

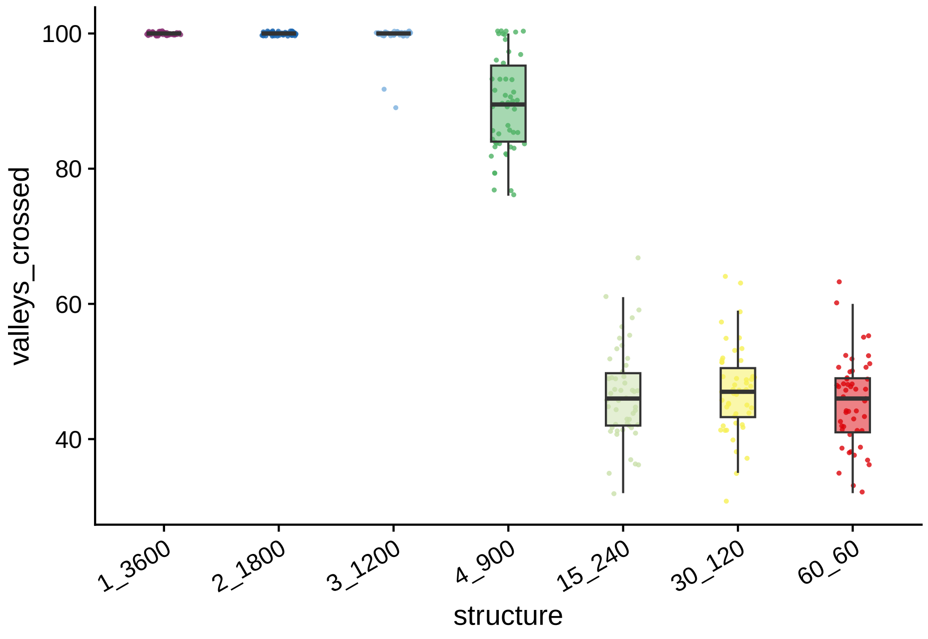
```
wc_results <- pairwise.wilcox.test(
  x = filter(max_org_data, landscape == "Multipath")$fitness,
  g = filter(max_org_data, landscape == "Multipath")$structure,
  p.adjust.method   = "holm",
  exact = FALSE
)

mp_fitness_wc_table <- kbl(wc_results$p.value) %>%
  kable_styling()

save_kable(
  mp_fitness_wc_table,
  paste0(plot_dir, "/multipath_fitness_wc_table.pdf")
)
mp_fitness_wc_table
```

5.2.3 Valleys crossed in valley-crossing landscape

```
valleycrossing_valleys_plt <- ggplot(
    data = filter(max_org_data, landscape == "Valley crossing"),
    mapping = aes(
      x = structure,
```

	1_3600	2_1800	3_1200	4_900	15_240	30_120
2_1800	1.0000000	NA	NA	NA	NA	NA
3_1200	0.0389539	0.2309342	NA	NA	NA	NA
4_900	0.0000552	0.0022081	0.6036094	NA	NA	NA
15_240	0.0000000	0.0000001	0.0000387	0.0022081	NA	NA
30_120	0.0000000	0.0000000	0.0000000	0.0000003	0.4456978	NA
60_60	0.0000000	0.0000000	0.0000002	0.0000094	0.6036094	1

```r
      y = valleys_crossed,
      fill = structure
    )
  ) +
  # geom_flat_violin(
  #   position = position_nudge(x = .2, y = 0),
  #   alpha = .8
  # ) +
  geom_point(
    mapping = aes(color = structure),
    position = position_jitter(width = .15),
    size = .5,
    alpha = 0.8
  ) +
  geom_boxplot(
    width = .3,
    outlier.shape = NA,
    alpha = 0.5
  ) +
  scale_color_discreterainbow() +
  scale_fill_discreterainbow() +
  theme(
    legend.position = "none",
    axis.text.x = element_text(
      angle = 30,
      hjust = 1
    )
  )
ggsave(
  filename = paste0(plot_dir, "/valleycrossing_valleys_crossed.pdf"),
  plot = valleycrossing_valleys_plt,
  width = 6,
  height = 4
)

valleycrossing_valleys_plt
```



```
vc <- max_org_data %>%
  filter(landscape == "Valley crossing") %>%
  group_by(structure) %>%
  summarize(
    reps = n(),
    median_valleys_crossed = median(valleys_crossed),
    mean_valleys_crossed = mean(valleys_crossed),
    min_valleys_crossed = min(valleys_crossed)
  ) %>%
  arrange(
    desc(mean_valleys_crossed)
  )
vc
```

```
## # A tibble: 7 x 5
##   structure  reps median_valleys_crossed mean_valleys_crossed
##   <fct>     <int>                  <dbl>                <dbl>
## 1 1_3600       50                  100                  100
## 2 2_1800       50                  100                  100
## 3 3_1200       50                  100                   99.6
## 4 4_900        50                   89.5                 89.3
## 5 30_120       50                   47                   47.2
## 6 15_240       50                   46                   46.6
## 7 60_60        50                   46                   45.5
## # i 1 more variable: min_valleys_crossed <dbl>
```

	1_3600	2_1800	3_1200	4_900	15_240	30_120
2_1800	NaN	NA	NA	NA	NA	NA
3_1200	0.796952	0.796952	NA	NA	NA	NA
4_900	0.000000	0.000000	0	NA	NA	NA
15_240	0.000000	0.000000	0	0	NA	NA
30_120	0.000000	0.000000	0	0	0.9787605	NA
60_60	0.000000	0.000000	0	0	0.9787605	0.796952

```
vc$min_valleys_crossed
```

```
## [1] 100 100  89  76  31  32  32
```

```
kruskal.test(
  formula = valleys_crossed ~ structure,
  data = filter(max_org_data, landscape == "Valley crossing")
)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  valleys_crossed by structure
## Kruskal-Wallis chi-squared = 309.49, df = 6, p-value < 2.2e-16
```

```
wc_results <- pairwise.wilcox.test(
  x = filter(max_org_data, landscape == "Valley crossing")$valleys_crossed,
  g = filter(max_org_data, landscape == "Valley crossing")$structure,
  p.adjust.method   = "holm",
  exact = FALSE
)

vc_valleys_crossed_wc_table <- kbl(wc_results$p.value) %>%
  kable_styling()

save_kable(
  vc_valleys_crossed_wc_table,
  paste0(plot_dir, "/valley_crossing_valleys_wc_table.pdf")
)
vc_valleys_crossed_wc_table
```