R Notebook

Running on Discovery

I recommend viewing this with a web-based Rstudio server on Discovery:

https://ood.discovery.neu.edu/pun/sys/dashboard/batch_connect/sys/RStudio/session_contexts/new

Press *Ctrl+Enter* to run a chunk.

Initialization

You may want to change the directory below.

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
knitr::opts_knit$set(root.dir = "/scratch/a.guha/minnpm-exp")
```

Load the data:

```
raw_data <- read_csv("results.csv",
  col_types = cols(Status=col_factor(),
                   Project=col_factor(),
                   Rosette=col_logical(),
                   Consistency=col_factor(),
                   Minimize=col_factor(),
                   Time=col_double(),
                   NDeps=col_integer()),
  show_col_types = FALSE)
```

Manual Verification Step

Check that these are the factors that appear below:

1. *success*: everything worked!
2. *ERESOLVE*: depends on something that isn't in the repository
3. *ETARGET*: requires some other target architecture **verify**
4. *EBADPLATFORM*: requires some other platform (e.g., macOS)
5. *EUNSUPPORTEDPROTOCOL:* a dependency is in a format that NPM does not support
6. *unexpected*: something went wrong on Discovery. See experiment.out

7. *unavailable*: something went wrong and we didn't even capture the result. See experiment.out

```
levels(raw_data$Status)
```

```
## [1] "success"              "ERESOLVE"             "ETARGET"
## [4] "EBADPLATFORM"         "EUNSUPPORTEDPROTOCOL" "unexpected"
## [7] "unavailable"
```

```
levels(raw_data$Consistency)
```

```
## [1] ""    "npm" "pip"
```

```
levels(raw_data$Minimize)
```

```
## [1] ""                           "min_oldness,min_num_deps"
## [3] "min_num_deps,min_oldness"   "min_duplicates,min_oldness"
## [5] "min_oldness,min_duplicates"
```

The value in the Count column should be 1000 for every row:

```
raw_data %>%
  group_by(Rosette,Minimize,Consistency) %>%
  summarize(Count = n())
```

```
## `summarise()` has grouped output by 'Rosette', 'Minimize'. You can override using the `.groups` argu
```

```
## # A tibble: 7 x 4
## # Groups:   Rosette, Minimize [5]
##   Rosette Minimize                     Consistency Count
##   <lgl>   <fct>                        <fct>       <int>
## 1 FALSE   ""                           ""           1000
## 2 TRUE    "min_oldness,min_num_deps"   "npm"        1000
## 3 TRUE    "min_oldness,min_num_deps"   "pip"        1000
## 4 TRUE    "min_num_deps,min_oldness"   "npm"        1000
## 5 TRUE    "min_num_deps,min_oldness"   "pip"        1000
## 6 TRUE    "min_duplicates,min_oldness" "npm"        1000
## 7 TRUE    "min_oldness,min_duplicates" "npm"        1000
```

Failures

How many failures occur for each configuration? We know we will see more failures than NPM, but hopefully not too many more.

```
raw_data %>%
  filter(Status != "success") %>%
  group_by(Rosette,Minimize,Consistency) %>%
  summarize(Count = n())
```

```
## `summarise()` has grouped output by 'Rosette', 'Minimize'. You can override using the `.groups` argu
```

```
## # A tibble: 7 x 4
## # Groups:   Rosette, Minimize [5]
##   Rosette Minimize                     Consistency Count
##   <lgl>   <fct>                        <fct>       <int>
## 1 FALSE   ""                           ""             38
## 2 TRUE    "min_oldness,min_num_deps"   "npm"          43
## 3 TRUE    "min_oldness,min_num_deps"   "pip"          77
## 4 TRUE    "min_num_deps,min_oldness"   "npm"          42
## 5 TRUE    "min_num_deps,min_oldness"   "pip"          77
```

```
## 6 TRUE    "min_duplicates,min_oldness" "npm"          45
## 7 TRUE    "min_oldness,min_duplicates" "npm"          43
```

Let's rule out failures that are due to unsolvability on our platform:

```
raw_data %>%
  filter(Status == "unexpected" | Status == "unavailable") %>%
  group_by(Rosette,Minimize,Consistency) %>%
  summarize(Count = n())
```

```
## `summarise()` has grouped output by 'Rosette', 'Minimize'. You can override using the `.groups` argu
```

```
## # A tibble: 7 x 4
## # Groups:   Rosette, Minimize [5]
##   Rosette Minimize                     Consistency Count
##   <lgl>   <fct>                        <fct>       <int>
## 1 FALSE   ""                           ""              1
## 2 TRUE    "min_oldness,min_num_deps"   "npm"          42
## 3 TRUE    "min_oldness,min_num_deps"   "pip"          76
## 4 TRUE    "min_num_deps,min_oldness"   "npm"          41
## 5 TRUE    "min_num_deps,min_oldness"   "pip"          76
## 6 TRUE    "min_duplicates,min_oldness" "npm"          44
## 7 TRUE    "min_oldness,min_duplicates" "npm"          42
```

These are likely due to timeouts, Z3 crashing, etc.

The Need for Tree-Solving

NPM uses a tree-solver because its easy to resolve conflicts. But, how many conflicts do you see with MinNPM in PIP mode?

TODO: Need to process output further to distinguish these errors.

Minimizing Number of Dependencies

For each project, the number of dependencies with vanilla NPM, and with MinNPM configured to minimize #deps and oldness, in that order.

```
min_dep_analysis <- bind_rows(raw_data %>%
            filter(Rosette == FALSE) %>%
            select(Project,NDeps) %>%
            mutate(Solver="NPM"),
          raw_data %>%
            filter(Rosette == TRUE & Consistency == "npm" &
                   Minimize == "min_num_deps,min_oldness") %>%
            select(Project, NDeps) %>%
            mutate(Solver="MinDeps")) %>%
  pivot_wider(values_from=NDeps, names_from=Solver)
```

In theory, MinNPM should always produce a smaller solution. But, it seems like it doesn't always.

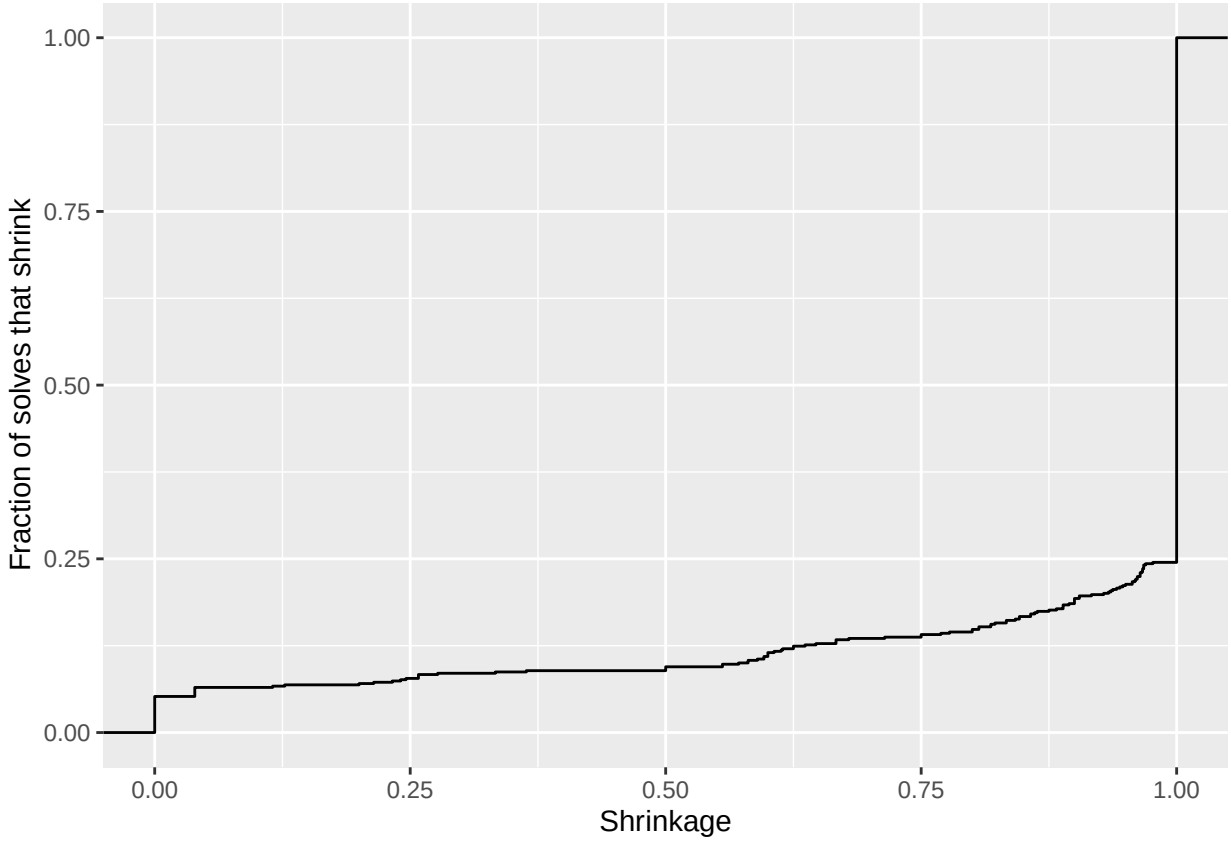
```
min_dep_analysis %>% mutate(Badness = MinDeps - NPM) %>% filter(Badness > 0)
```

```
## # A tibble: 17 x 4
##    Project                NPM MinDeps Badness
##    <fct>                <int>   <int>   <int>
##  1 https-proxy-agent        0       3       3
```

```
##  2 jest-matcher-utils        0      20      20
##  3 jest-haste-map            0      35      35
##  4 request                   0      40      40
##  5 jest-diff                 0      19      19
##  6 agent-base                0       2       2
##  7 saxes                     0       1       1
##  8 terser-webpack-plugin     0      25      25
##  9 http-proxy-agent          0       4       4
## 10 jsdom                     0      58      58
## 11 tsutils                   0       1       1
## 12 babel-loader              0      23      23
## 13 ts-node                   0      13      13
## 14 eslint-utils              0       1       1
## 15 babel-runtime             2      25      23
## 16 rxjs                      0       1       1
## 17 fast-levenshtein          0       1       1
```

The graph below is bogus, since I've filtered out the outliers.

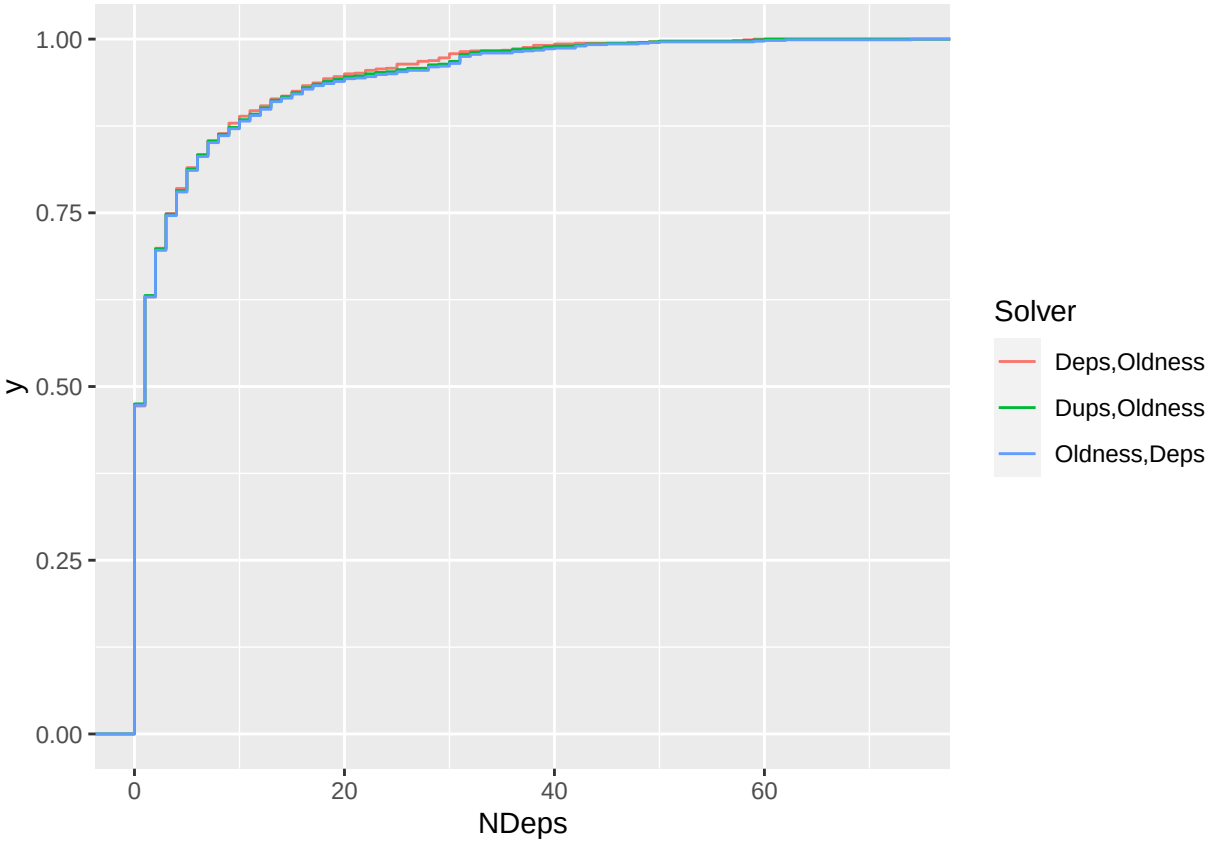
```
min_dep_analysis %>%
  mutate(Shrinkage = MinDeps / NPM) %>%
  filter(Shrinkage <= 1.0) %>%
  select(Shrinkage) %>%
  ggplot(aes(Shrinkage)) +
  stat_ecdf() +
  ylab("Fraction of solves that shrink")
```



Impact on Priorities

Skip this? Not very informative.

```
bind_rows(
  raw_data %>%
    filter(Rosette == TRUE & Consistency == "npm" &
             Minimize == "min_num_deps,min_oldness") %>%
    select(Project,NDeps) %>%
    mutate(Solver="Deps,Oldness"),
  raw_data %>%
    filter(Rosette == TRUE & Consistency == "npm" &
             Minimize == "min_oldness,min_duplicates") %>%
    select(Project,NDeps) %>%
    mutate(Solver="Oldness,Deps"),
  raw_data %>%
    filter(Rosette == TRUE & Consistency == "npm" &
             Minimize == "min_duplicates,min_oldness") %>%
    select(Project, NDeps) %>%
    mutate(Solver="Dups,Oldness")) %>%
  ggplot(aes(NDeps,color=Solver)) +
  stat_ecdf()
```



Slowdown of MinNPM

These were run on different kinds of machines, etc. and in parallel. So timing is not reliable. But, we can assume the trends hold, and we have seen these trends over lots of experiment runs. Some questions:

1. Why is it that the Rosette solver is faster? It seems almost unbelievable that the NPM solver is slower to me.
2. Should we somehow represent the timeouts on this graph?

```
bind_rows(
  raw_data %>%
    filter(Rosette == FALSE) %>%
    select(Project,Time) %>%
    mutate(Solver="NPM"),
  raw_data %>%
    filter(Rosette == TRUE & Consistency == "npm" &
             Minimize == "min_num_deps,min_oldness") %>%
    select(Project,Time) %>%
    mutate(Solver="MinNPM")) %>%
  pivot_wider(values_from=Time, names_from=Solver) %>%
  mutate(DeltaTime = MinNPM- NPM) %>%
  select(-NPM, -MinNPM) %>%
  ggplot(aes(y=DeltaTime)) +
  stat_bin()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 66 rows containing non-finite values (stat_bin).
```