Reproducing Rq1 and Rq3

Jeffrey M. Young

15 May 2020

Hello, in this walkthrough we'll be the data analysis and generating the plots for rq1, and rq3.

First the usual invocations for libraries, note that I have silenced the shadowing warnings from R:

```
library(ggplot2) #plotting
library(dplyr)   #dataframe manipulation
library(tidyr)   #dataframe manipulation
library(broom)   #for the tidy function
library(scales)  #for scientific function
```

Now let's load the data, we immediately pipe this with dplyr for conveniences like manipulating the arrow in v–>v to look nice and making factors. The Name column is dropped because it is large and separated to the other columns. We use the name `Config` to stand for version variants throughout the scripts.

```
## Pound signs are comments
## leftarrow (<-) is assignment, `=` also is assignment but their meanings subtly differ
## <- is the accepted default
finResultsFile  <- "../../data/fin_data.csv"
autoResultsFile <- "../../data/auto_data.csv"
finRawFile      <- "../../data/fin_rq3_singletons.csv"
autoRawFile     <- "../../data/auto_rq3_singletons.csv"

finData <- read.csv(file=finResultsFile) %>%
  ## %>% pipes, similar to `|` in Unix, (.) in Haskell, and |> in Julia
  mutate(Algorithm = as.factor(Algorithm), Config = as.factor(Config)) %>%
  mutate(Algorithm = gsub("-->", "\U27f6", Algorithm)) %>% select(-Name)

autoData <- read.csv(file=autoResultsFile) %>%
  ## mutate changes a column on the left hand side of = by the RHS
  mutate(Algorithm = as.factor(Algorithm), Config = as.factor(Config)) %>%
  mutate(Algorithm = gsub("-->", "\U27f6", Algorithm)) %>% select(-Name)

### we can also add columns with mutate, _and_ keep the rest of the data frame
### in this example we add the `data` column for each dataset
finDF <- finData %>% mutate(data = "Fin")
autoDF <- autoData %>% mutate(data = "Auto")

data <- rbind(finDF, autoDF)

### show the structure of the data
str(data)
```

```
## 'data.frame':    112 obs. of  17 variables:
##  $ Mean            : num  1.01 1.169 0.813 0.801 1.091 ...
```

```
##  $ MeanLB          : num  0.96 1.12 0.781 0.768 1.03 ...
##  $ MeanUB          : num  1.065 1.218 0.854 0.848 1.165 ...
##  $ Stddev          : num  0.0871 0.0803 0.0633 0.0629 0.1139 ...
##  $ StddevLB        : num  0.0633 0.0616 0.0439 0.0329 0.071 ...
##  $ StddevUB        : num  0.1334 0.1126 0.0883 0.0974 0.1708 ...
##  $ DataSet         : Factor w/ 1 level "Z3": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Algorithm       : chr  "v v" "v v" "v v" "v v" ...
##  $ Config          : Factor w/ 19 levels "V1","V1*V2","V1*V2*V3",..: 1 12 13 14 15 16 17 18 19 11 ..
##  $ ChcCount        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PlainCount      : int  7803 9953 5070 5398 9525 9418 5872 5983 5589 5152 ...
##  $ CompressionRatio: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ VCoreSize       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ VCorePlain      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ VCoreVar        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Variants        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ data            : chr  "Fin" "Fin" "Fin" "Fin" ...
```

RQ1: Performance as variant count increases

Most of this should be explanatory if you are referencing the paper as well. The majority of the statistics such as `stddevLB` are not used but included for the interested. For each research question we construct the proper data frame, e.g., `rq1DF` by mutating or filtering `data`:

```
rq1DF <- data %>%
  filter(Variants >= 2) %>%
  group_by(Algorithm) %>%
  arrange(Variants)
```

We filter here because `data` includes singleton times as well. A singleton, in this analysis, is the result data entry where a version variant was solved as a plain formula. For rq1, we need to filter out the data where `Variants == 2`, which correspond to VPL formulas which solved only a single version variant. This is in contrast with the data which composes rq3, for rq3, we transformed all version variant formulas to plain propositional formulas to assess overhead. Thus, these formulas have `Variants == 0` because they are plain. Note that for most analyses we'll be using the `group_by` verb to not mix the datasets. Now we can plot rq1 with `ggplot`:

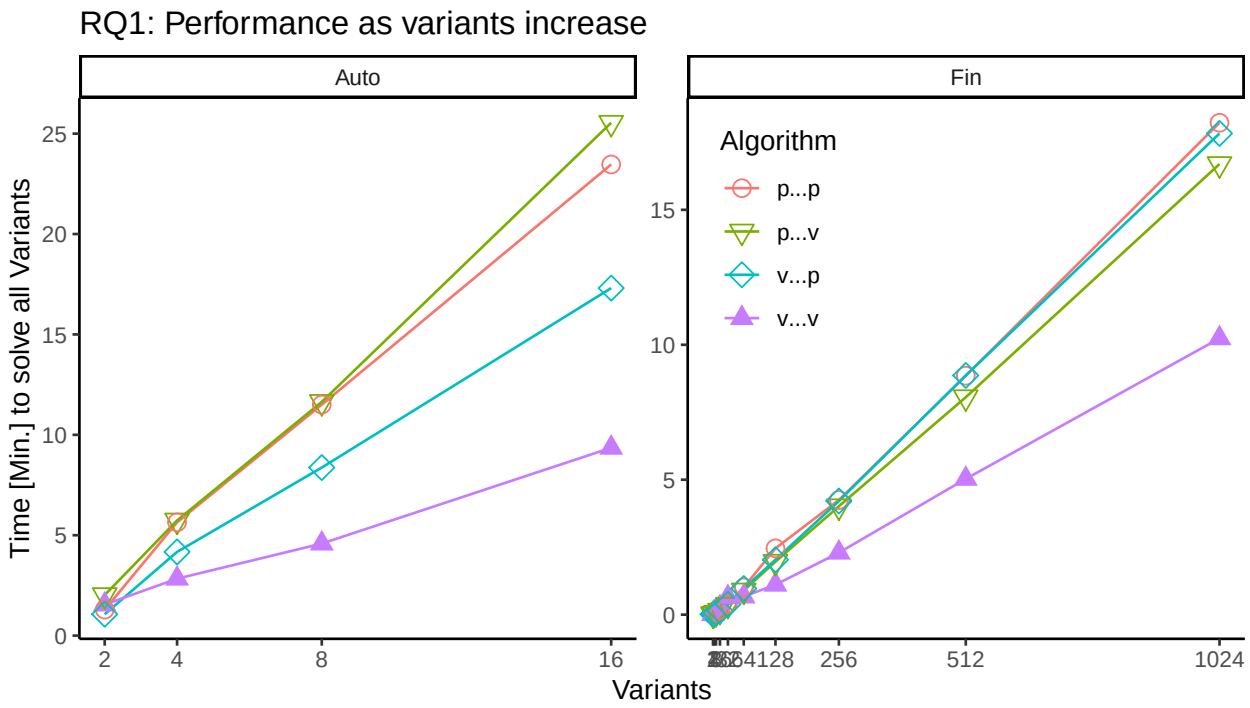
```
## custom breaks on the x-axis for each dataset
breaksRq1 <- function(x) {
  if (max(x) > 16) {
    2^(1:10)
  } else {
    2^(1:4)}
  }

## we use Hadley Wickham's ggplot for plotting, the first line invokes a ggplot object
rq1 <- ggplot(rq1DF) +
  ## we add lines and a point to the plot
  geom_line(aes(x=Variants, y=Mean/60, color=Algorithm)) +
  geom_point(aes(x=Variants, y=Mean/60, shape=Algorithm, color=Algorithm),size=3) +
  ## then we scale the values manually so algorithms have the same shape in each plot
  scale_shape_manual(values = c(1,6,5,17)) +
  ## change the x-axis tick marks according to our breaking function
  scale_x_continuous(breaks=breaksRq1, limits=c(2,NA)) +
  ## create sub plots by the `data` column
  facet_wrap(. ~ data, scales = "free") +
```

```
  ## a clean publication ready theme
  theme_classic() +
  ## titles
  ggtitle("RQ1: Performance as variants increase") +
  ylab("Time [Min.] to solve all Variants") +
  ## change the legend position
  theme(legend.position = c(0.6,0.75))
```

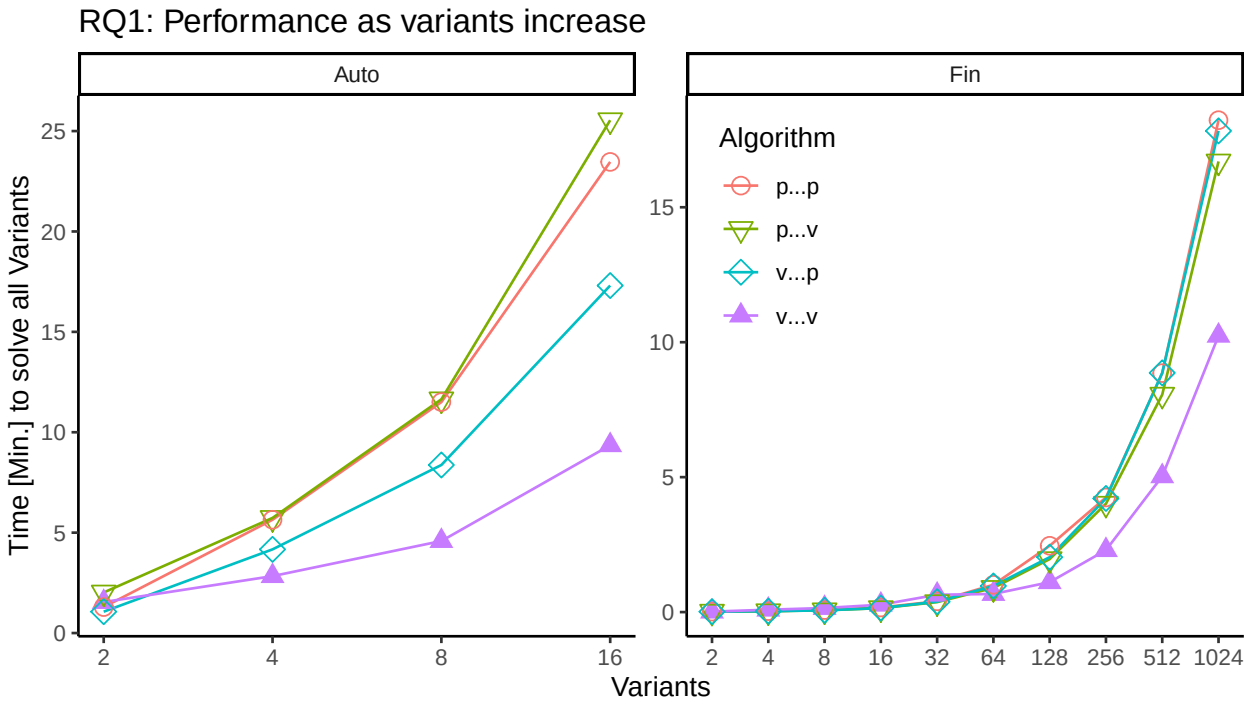
Most of this is fine tuning for a nice plot. The plot is just a line plot with points for emphasis. We use the clean classic theme, and move the legend manually to render well with LaTeX. We scale the y-axis to minutes and manually select shapes for the algorithms so that each algorithm uses the same shape across all of the plots. Finally we facet by dataset. Rmarkdown has problems with the algorithm arrow so I have muted the warnings. This is the publication quality plot, sans the arrows of course:

```
rq1
```



And here is the same plot with a log scale x-axis so you can see data at lower variant count:

```
ggplot(rq1DF) +
  geom_line(aes(x=Variants, y=Mean/60, color=Algorithm)) +
  geom_point(aes(x=Variants, y=Mean/60, shape=Algorithm, color=Algorithm),size=3) +
  scale_shape_manual(values = c(1,6,5,17)) +
  scale_x_log10(breaks=breaksRq1, limits=c(2,NA)) +
  facet_wrap(. ~ data, scales = "free") +
  theme_classic() +
  ggtitle("RQ1: Performance as variants increase") +
  ylab("Time [Min.] to solve all Variants") +
  theme(legend.position = c(0.6,0.75))
```



RQ3: Overhead of variational solving

For this analysis we need to filter data for the complement data set of rq1:

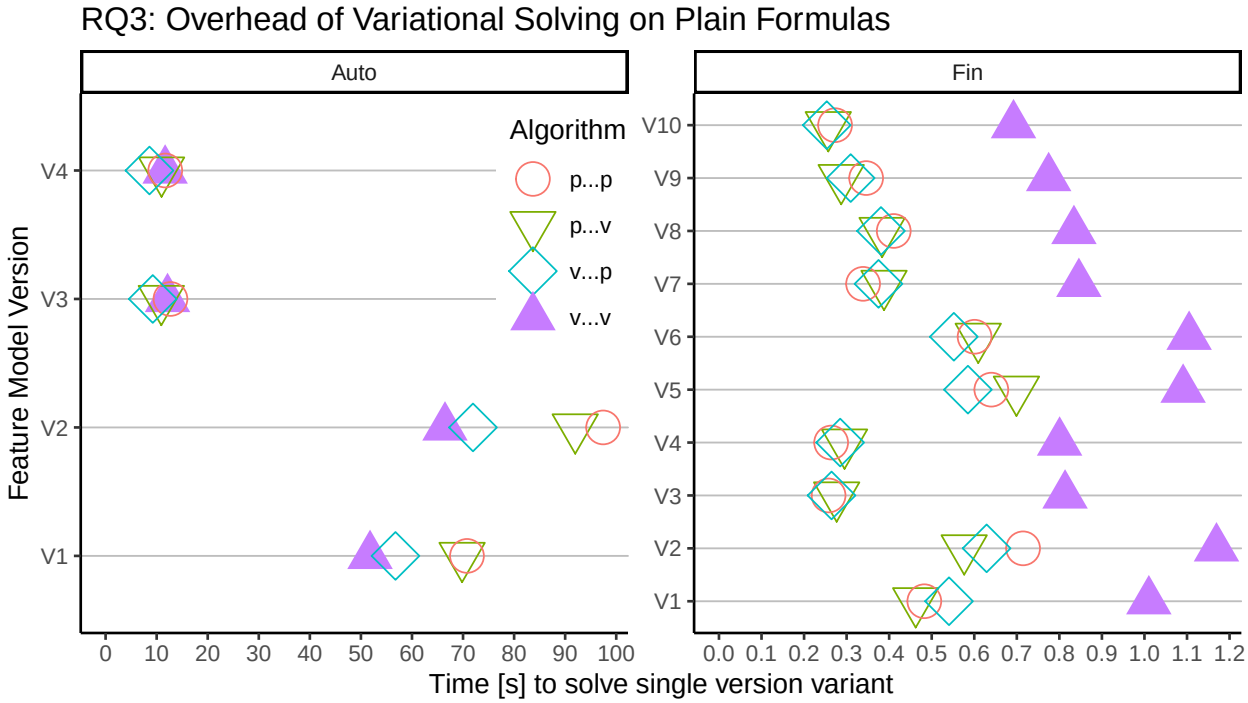
```
rq3DF <- data %>% filter(ChcCount == 0) %>%
  mutate(plotOrdering = as.numeric(substring(Config, 2))) %>%
  mutate(Config = factor(Config, levels = c("V1", "V2", "V3", "V4", "V5", "V6",
                                            "V7", "V8", "V9", "V10")))
```

Again, we just add some niceties to order the versions in the `rq3` plot and remove all data that was run on variational formulas. Now to generate the plot we use a custom breaking function again, to have the proper axis scale for each sub-plot (or facet in R terminology):

```
## custom breaks for the facets
breaksRq3 <- function(x) {
  if (max(x) < 4) {
    ## then we are in fin, NA to just have R find the max
    seq(0, 1.2, 0.10)
  } else {
    seq(0, 150, 10)
  }
}

ggplot(rq3DF, aes(x=Config, y=Mean, fill=Algorithm, shape=Algorithm, color=Algorithm)) +
  geom_point(size=6) +
  scale_shape_manual(values = c(1,6,5,17)) +
  theme_classic() +
  scale_y_continuous(limits=c(0, NA), breaks=breaksRq3) +
  facet_wrap(.~ data, scales="free") +
  ggtitle("RQ3: Overhead of Variational Solving on Plain Formulas") +
  ylab("Time [s] to solve single version variant") +
  xlab("Feature Model Version") +
```

```
  theme(legend.position = c(0.42,0.75),
        legend.key.size = unit(.65,'cm')) +
  theme(panel.grid.major.y = element_line(color = "grey")) +
  coord_flip()
```



Note that we `coord_flip` to put versions on the y-axis. We chose this format to show lots of comparisons succinctly. We can also calculate the exact slowdowns with the dataset now:

```
rq3DF %>% group_by(data,Algorithm) %>%  summarise(AvgMean = mean(Mean))
```

```
## # A tibble: 8 x 3
## # Groups:   data [2]
##   data  Algorithm AvgMean
##   <chr> <chr>        <dbl>
## 1 Auto  p p         48.1
## 2 Auto  p v         45.9
## 3 Auto  v p         36.6
## 4 Auto  v v         35.5
## 5 Fin   p p          0.433
## 6 Fin   p v          0.423
## 7 Fin   v p          0.418
## 8 Fin   v v          0.914
```

but a box plot or something similar of the raw values will give a clearer picture of the distributions. We'll reuse this data for the two way anova:

```
finSingData <- read.csv(file=finRawFile) %>%
  mutate(Algorithm = as.factor(Algorithm), Config = as.factor(Config)) %>%
  mutate(Algorithm = gsub("-->", "\U27f6", Algorithm), data = "Financial") %>%
  group_by(Algorithm, Config) %>%
  mutate(TimeCalc = time -append(0,head(time, -1))) %>% filter(TimeCalc > 0)
```

```
autoSingData <- read.csv(file=autoRawFile) %>%
  mutate(Algorithm = as.factor(Algorithm), Config = as.factor(Config)) %>%
  mutate(Algorithm = gsub("-->", "\U27f6", Algorithm), data = "Auto") %>%
  group_by(Algorithm, Config) %>%
  mutate(TimeCalc = time -append(0,head(time, -1))) %>% filter(TimeCalc > 0)

### combine the data sets
singData <- rbind(finSingData, autoSingData)
```

For the raw data we use a special function to calculate the offset time the benchmark library `gauge` returned. This is essentially a `zipWith (-) l1 l2` where l2 is shifted l1 by a single element. Let's peek at the data:

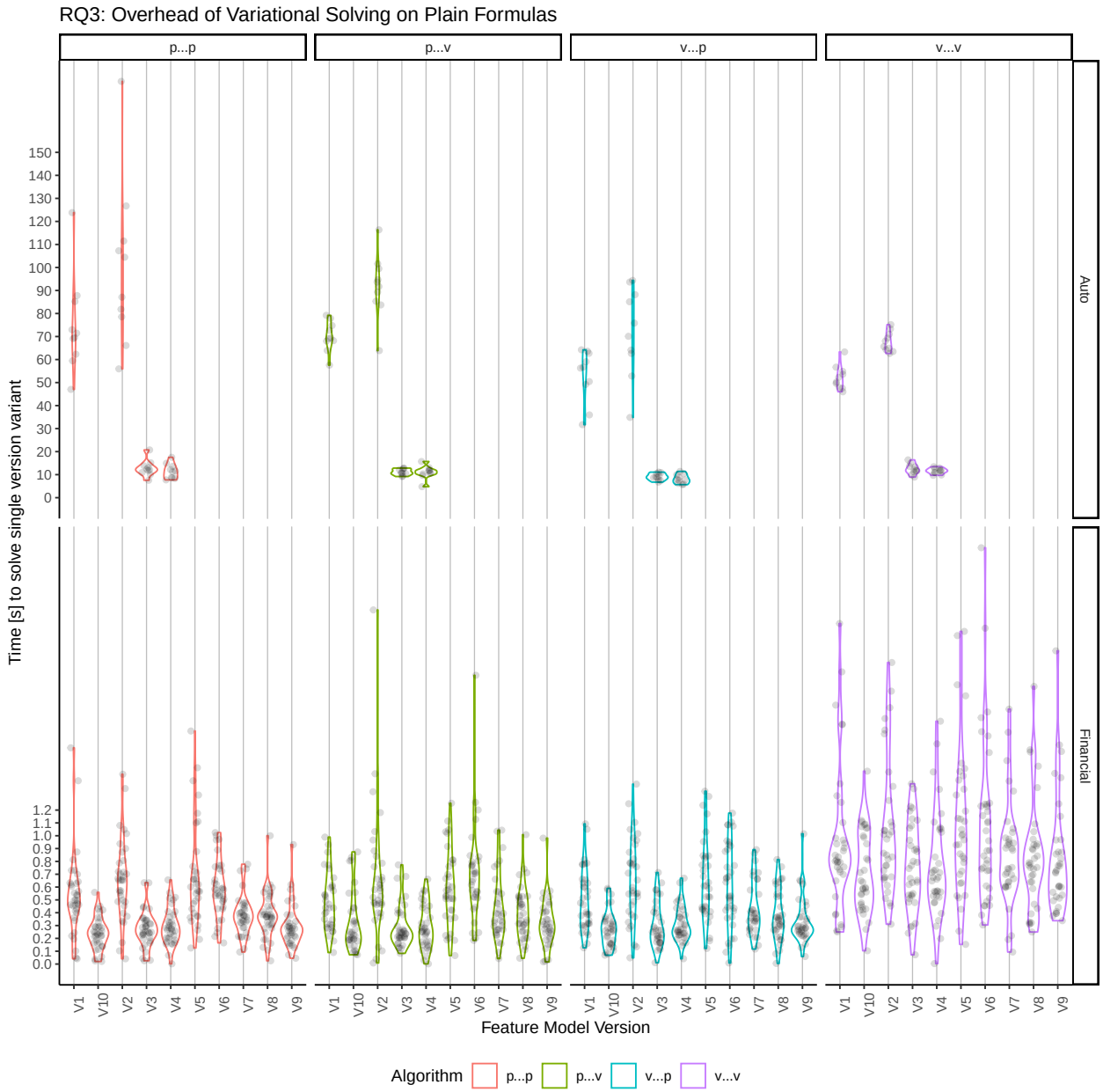
```
peek <- singData %>%
  select(data,Config,Algorithm,TimeCalc) %>%
  group_by(data, Config, Algorithm) %>%
  summarise(Mean = mean(TimeCalc), Stddev = sd(TimeCalc))

peek %>% head
```

```
## # A tibble: 6 x 5
## # Groups:   data, Config [2]
##   data  Config Algorithm  Mean Stddev
##   <chr> <chr>  <chr>     <dbl>  <dbl>
## 1 Auto  V1     p p        74.9  20.9
## 2 Auto  V1     p v        70.1   6.47
## 3 Auto  V1     v p        53.0  11.3
## 4 Auto  V1     v v        52.2   5.19
## 5 Auto  V2     p p       100.   35.7
## 6 Auto  V2     p v        92.0  13.6
```

We can see there is a lot of variance in some of the samples. We can visualize the distribution with a violin plot for each config, facet by algorithm to see a matrix of sub plots that shows the sample distribution of each version variant (Config) and each algorithm:

```
ggplot(singData, aes(x=Config, y=TimeCalc)) +
  geom_violin(aes(color=Algorithm)) +
  geom_jitter(width=0.2, alpha=0.15) +
  theme_classic() +
  scale_y_continuous(limits=c(0, NA), breaks=breaksRq3) +
  facet_grid(data ~ Algorithm, scales="free") +
  ggtitle("RQ3: Overhead of Variational Solving on Plain Formulas") +
  ylab("Time [s] to solve single version variant") +
  xlab("Feature Model Version") +
  theme(panel.grid.major.x = element_line(color = "grey"),
  legend.position="bottom", axis.text.x=element_text(angle=90))
```



Notice the amount of variance (height of the violins) which exists in raw data, and clearly that `v-->v` has a slowdown for `financial`. Let's confirm the differences between algorithms and groups with a two-way ANOVA:

RQ3: Statistical Comparisons

We demonstrate the two-way ANOVA on `financial` and show the normality assumption is violated. This analysis is in the bottom of `rq1_rq3.R` and `sigTest.R`. We'll perform the analysis on `financial` first and leave `auto` for the interested reader (who can also simply uncomment the code in the aforementioned R scripts if they so choose):

First we perform the ANOVA on the raw data

```
### Do a two-way anova looking at both Algorithm, Config and their interaction
res.fin.aov <- aov(TimeCalc ~ Algorithm * Config, data = finSingData)
```

Here we tell R that we want to test the dependent variable TimeCalc as a function of two independent variables Algorithm and Config, *and* the interaction Algorithm:Config, i.e., TimeCalc ~ Algorithm + Config + Algorithm:Config. Let's have a look at the model:

```
res.fin.aov
```

```
## Call:
##    aov(formula = TimeCalc ~ Algorithm * Config, data = finSingData)
##
## Terms:
##                  Algorithm    Config Algorithm:Config Residuals
## Sum of Squares   53.82664  28.43916          0.88739 140.42716
## Deg. of Freedom         3         9               27      1349
##
## Residual standard error: 0.322641
## Estimated effects may be unbalanced
```

It is strange that the sum of squares reduces in magnitude by an order of magnitude for the Algorithm:Config interaction. Now we check to see which combinations are significant, R automatically calculates an adjusted p-value to control for family-wise error rate (error introduced by performing many tests):

```
### Check the summary to see what is significant
summary(res.fin.aov)
```

```
##                    Df Sum Sq Mean Sq F value Pr(>F)
## Algorithm           3  53.83  17.942 172.360 <2e-16 ***
## Config              9  28.44   3.160  30.355 <2e-16 ***
## Algorithm:Config   27   0.89   0.033   0.316      1
## Residuals        1349 140.43   0.104
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that both Algorithm and Config are significant. We expect at least Config to be significant because different SAT problems are likely to have different run-times. It is strange that two independent variables are significant, but not their interaction is not, especially for the magnitude of f-values being reported, and for the nature the object of analysis (i.e., a machine, not a biological system). Next we perform the Tukey-pairwise comparison to see *exactly* which combinations are significant.

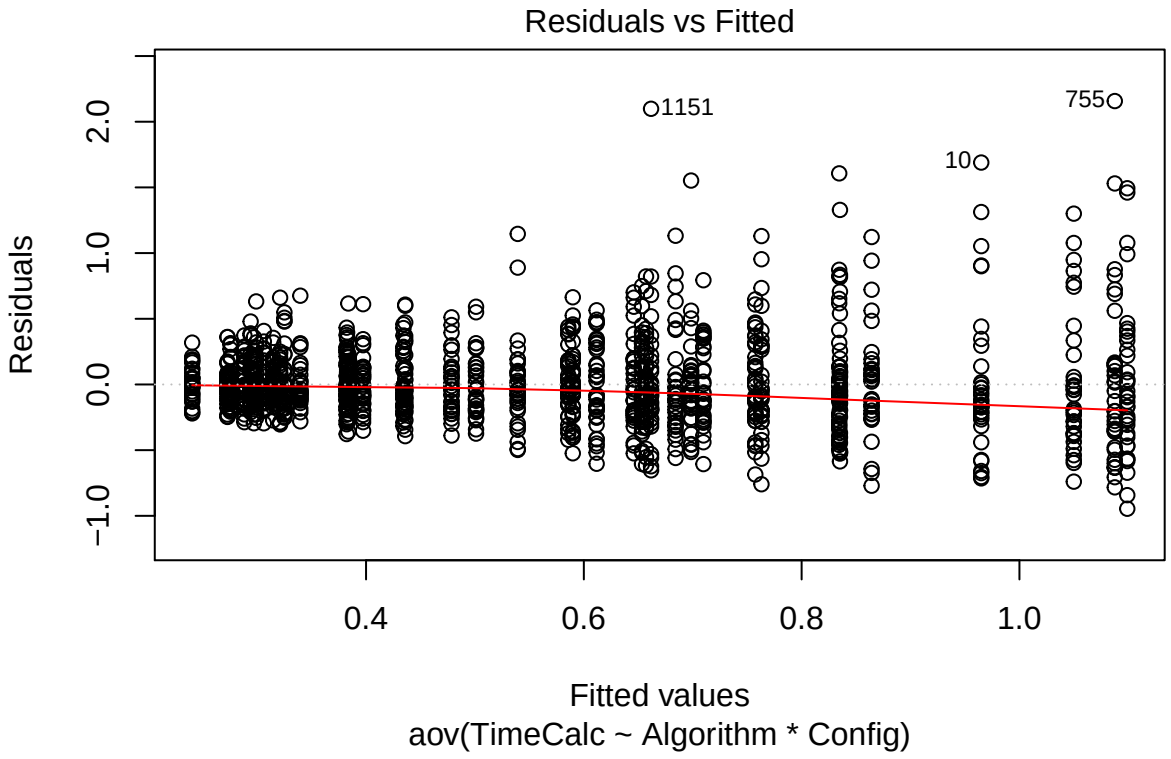
```
## perform the analysis
TukeyHSD(res.fin.aov, which = "Algorithm:Config") %>%
  tidy %>%
  ### cleanup
  separate(comparison, sep=c(3, 4, 7, 11)
         , into = c("AlgLeft", "Dump", "ConfigLeft", "AlgRight", "ConfigRight")) %>%
  mutate(ConfigLeft = gsub("-", "", ConfigLeft),
         ConfigRight = gsub(":","", ConfigRight),
         AlgRight = gsub(":|-", "", AlgRight),
         data = "Financial",
         pVal = scientific(adj.p.value, 3)) %>%
  select(-Dump) %>%
  ## remove out-group comparisons between different variants, e.g., V1 compared with V10
  filter(ConfigLeft == ConfigRight) %>%
  mutate(Comparison=paste(AlgLeft,":",AlgRight,":",ConfigLeft, sep=""),
         AlgComparison=paste(AlgLeft,":",AlgRight, sep="")) %>%
  ## sort by p-value
  arrange(adj.p.value) %>%
```

```
  ## drop some columns to see p-value
  select(-term,-conf.low,-conf.high) %>%
  ## filter non-significant combinations
  filter(adj.p.value <= 0.05)
```

```
## # A tibble: 30 x 10
##    AlgLeft ConfigLeft AlgRight ConfigRight estimate adj.p.value data  pVal
##    <chr>   <chr>      <chr>    <chr>          <dbl>       <dbl> <chr> <chr>
##  1 v v     V9         p p      V9             0.535     6.57e-9 Fina~ 6.57~
##  2 v v     V5         p v      V5             0.509     4.51e-8 Fina~ 4.51~
##  3 v v     V9         p v      V9             0.513     5.89e-8 Fina~ 5.89~
##  4 v v     V6         p p      V6             0.502     6.24e-8 Fina~ 6.24~
##  5 v v     V3         p p      V3             0.485     2.60e-7 Fina~ 2.60~
##  6 v v     V9         v p      V9             0.495     3.59e-7 Fina~ 3.59~
##  7 v v     V1         p v      V1             0.486     3.97e-7 Fina~ 3.97~
##  8 v v     V6         v p      V6             0.476     4.07e-7 Fina~ 4.07~
##  9 v v     V4         p v      V4             0.475     7.54e-7 Fina~ 7.54~
## 10 v v     V7         p p      V7             0.482     7.61e-7 Fina~ 7.61~
## # ... with 20 more rows, and 2 more variables: Comparison <chr>,
## #   AlgComparison <chr>
```

We perform the test for each interaction, then cleanup the resulting data frame by separating out the exact comparisons into new columns, then filtering comparisons between version variants, e.g., clearly comparing V_1 with V_2 is meaningless. Then we sort by `adj.p.value`, just with a cursory glance we can see there are many significant comparisons. Thus, before digging into the comparisons we check the ANOVA assumptions, outliers, and residuals:

```
### Check leverage of the observations for outliers, this is in base R not ggplot2
plot(res.fin.aov, 1)
```



We can see that we have three outliers: 1151, 10, and 755. If we had homogenous variance in the dataset we

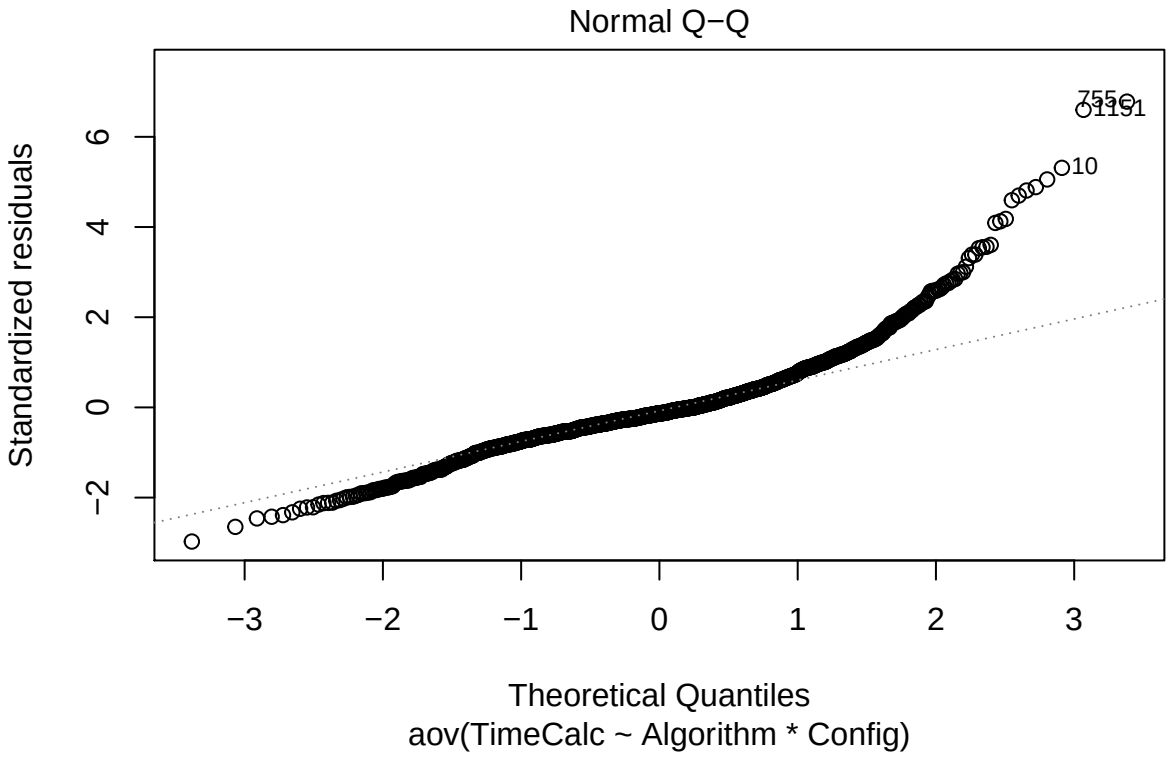
would expect the redline to remain horizontal, instead we observe the variance changing after ~0.5. We can formally check the homogeneity of variance with a `levene's test`:

```
library(car)
leveneTest(TimeCalc ~ Algorithm * Config, data=finSingData)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##         Df F value    Pr(>F)
## group   39  6.1567 < 2.2e-16 ***
##       1349
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

With a `p-value` of `< 0.05` from the `Levene's test` we have confirmed variance in this data set is non-homogenous. Next we'll check the normality assumption of the ANOVA, if we have a normal distribution between groups (e.g., Algorithms and Configs) then the residuals of the model will follow an approximate straight line in this plot. That is, we should observe a linear trend, any curving is indicative of a non-normal distribution:

```
### Check the normality assumption of the ANOVA
plot(res.fin.aov, 2)
```



Again, we observe curvation, indicative of non-normality at extreme quantiles. This is likely indicative of performance hitting a resource limit of some kind, although more analysis would be required on the tool to directly assess the signal. Clearly we do not have a normal distribution within groups here, we can formally check this with a Shapiro-Wilks test:

```
shapiro.test(x = residuals(res.fin.aov))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(res.fin.aov)
```

```
## W = 0.90245, p-value < 2.2e-16
```

And we see that the `p-value` *is significant* which indicates we have non-normal distributions. We leave the same analysis for `auto` up to the interested reader. For the analysis using the Kruskall-Wallis test please see the `sigTest` walkthrough.