ACOTSP 2000: Estimation of parameter importance with Random Forests (Performance)

Leslie Pérez Cáceres

In this document we are testing how to use Random Forest to assess importance and interactions of parameters based on the data gathered by irace.

For the analysis below we use as example ACOTSP:

- 20 secs cut off time
- 11 parameters
- 200 instances of size 2000
- 5000 experiments for configuration

We use random forest for predicting

1. configuration performance
2. configuration normalized performance
3. configuration performance quartile

Models are trained using default settings of the package Random Forest, excepting the number of trees that was set to 300. We have access to the following measures that can be considered indicators of importance:

- mean_min_depth: mean depth of the subtree closest to the tree root, where a variable is root of the sub tree.
- no_of_nodes: number of nodes in which the variable was used to split
- mse_increase: increment of prediction mean squared error
- no_of_trees: number of trees in which the variable was used
- times_a_root: number of times a variable was selected as root variable
- accuracy_decrease: measure of the classification accuracy (only for classification models)

For this analysis we use data generated by irace, it is possible to select a subset of the data and perform the analysis. The data is imputed and there is a procedure to analyze importance based on a reference variable and a retraining scheme to assess real importance in conditional parameters. The instance is also used as a predictor in this data. (details in another document)

There are some things that should be investigated regarding the best way to use the models to asses interaction and importance:

1. Discretising numerical variables for prediction: based on a comment I got that RF are biased to select numerical variables as split. This will be particularly interesting and in line with the fact that irace defines a sampling range around the current value.

2. Adjusting the number of trees and depth of them. My intuition is that smaller more smaller trees would be more useful in the task of detecting importance. This is because lower level splits are not as interesting and higher level splits. Also, to be used as a post-execution analysis tool the execution time required to build the model depends on these parameters.

3. How to understand how this importance or interaction is materialized i.e., which are the best parameter values ans how these interact. Can we have an heuristic idea of this interpreting the forest splits?

4. How to interpret instance importance and interaction when using models not predicting directly the performance. Can this help detecting heterogeneous sets?

In the following examples the ACOTSP data is used to predict **solution quality similarly** of how it is used in SMAC and GGA++. The intuition is that in these models instance is probably the best predictor of the quality due to the different sizes that compose the instance set and the nature of the configuration objective (tour length). In this model all irace data execution was used for training. Note that due to the focused n ature of irace data (given model convergence) there might be contradicting or not consistent interactions in the data.

1. Configuration performance as dependent variable

The data set used to train this model defines the variable to predict as the raw performance.

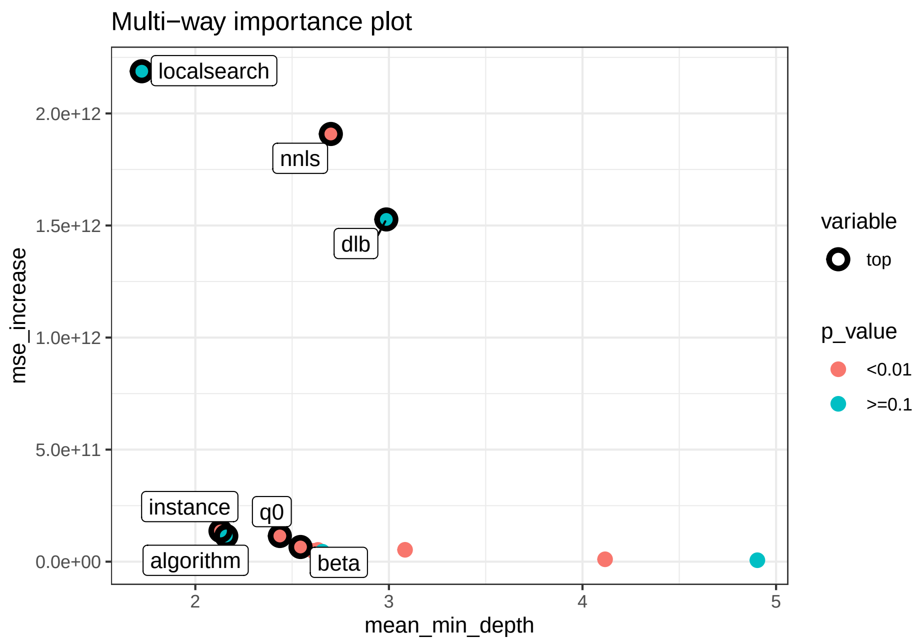
We show importance measures ordered by the mean min depth given that the interaction analysis is based on this measure.

```r
load("../model_data/model-acotsp2000-performance.Rdata")
importance_frame = model$importance_frame
important_parameters = model$important_parameters
full_interactions_frame = model$full_interactions_frame
interactions_frame = model$interactions_frame
kable(importance_frame[order(importance_frame[,"mean_min_depth"]),])  %>%
      kable_styling(latex_options="scale_down")
```

	variable	mean_min_depth	no_of_nodes	mse_increase	node_purity_increase	no_of_trees	times_a_root	p_value
9	localsearch	1.723333	7646	2.188235e+12	5.925592e+15	300	76	1
8	instance	2.130000	57488	1.369469e+11	4.592866e+14	300	8	0
1	algorithm	2.160000	8952	1.147517e+11	5.235803e+14	300	32	1
11	q0	2.436667	35931	1.146409e+11	5.453561e+14	300	35	0
4	beta	2.543333	57303	6.552375e+10	1.567375e+14	300	0	0
2	alpha	2.606667	59208	4.839387e+10	1.321584e+14	300	0	0
3	ants	2.633333	52801	5.253090e+10	1.946507e+14	300	17	0
7	elitistants	2.656667	4206	4.434819e+10	1.270095e+14	300	2	1
10	nnls	2.700000	46114	1.908346e+12	5.110555e+15	300	65	0
5	dlb	2.986667	6138	1.527083e+12	4.281041e+15	300	54	1
13	rho	3.083333	53372	5.296427e+10	1.461576e+14	300	11	0
6	dummy	4.116667	41433	1.063331e+10	2.827024e+13	300	0	0
12	rasrank	4.903333	6276	6.438292e+09	1.846431e+13	300	0	1

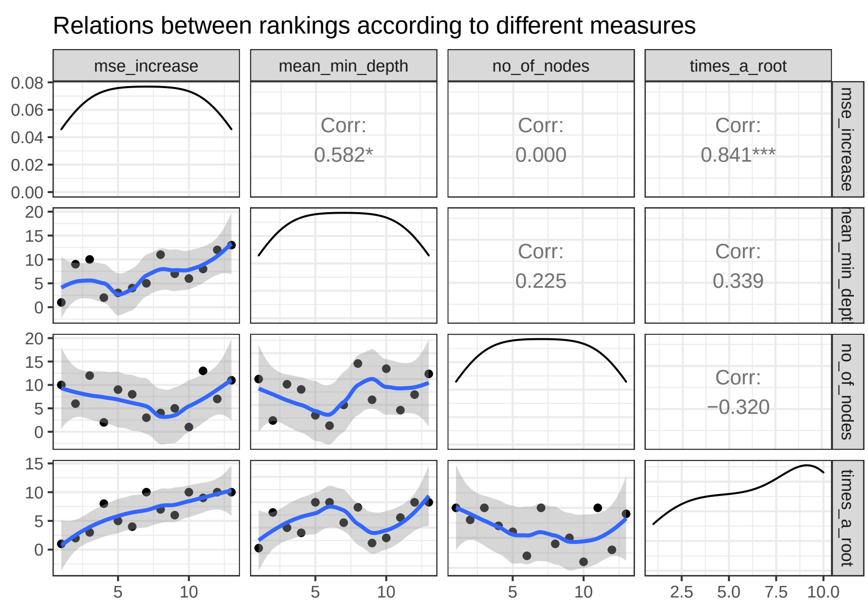
We can plot two importance measures using the randomForestExplainer package, we choose the to show the mean min depth in the x axis and the mse increase in the y axis. Top 7 variables are highlighted. In this case, since we are not predicting performance the effect of instance variable in model performance must be interpreted carefully. These plots are interesting given that contrast importance for ranking prediction in terms of accuracy (mse_increase) and in terms of early importance the tree more in line with classification goals (mean min depth).

```r
suppressWarnings(plot_multi_way_importance(importance_frame, size_measure = "p_value",
                          y_measure="mse_increase", x_measure="mean_min_depth",
                          no_of_labels=7))
```



We can also visualize the relationship between importance measures, this could help to understand which indicator is more suited or can be used as a complement of other:

```
plot_importance_rankings(importance_frame, measures=c("mse_increase", "mean_min_depth",
                                                      "no_of_nodes", "times_a_root"))
```



We perform an analysis of conditional parameters importance using a filtering and re training strategy and apply an irrelevant parameter filter by including a reference parameter. After this process the most important 5 parameters are detected.

Important parameters:

```
print(important_parameters)
```

```
## [1] "localsearch" "algorithm"   "instance"    "dummy"
```

Next, we run the interaction analysis only over important parameters. This is assuming the interactions one cares to detect are related to them, which might be not entirely correct ans should be evaluated as heuristic. The importance of interactions is calculated based on the mean min depth indicator in its conditional version.

Parameter interaction importance:

```
kable(full_interactions_frame)  %>%  kable_styling(latex_options="scale_down")
```

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth
instance	instance	0.6996466	283	instance:instance	1.410000
instance	algorithm	0.8912132	281	algorithm:instance	1.410000
instance	dummy	0.8222968	280	dummy:instance	1.410000
instance	localsearch	0.9884099	277	localsearch:instance	1.410000
dummy	dummy	1.1471967	276	dummy:dummy	1.473333
algorithm	algorithm	1.5899647	274	algorithm:algorithm	1.400000
localsearch	localsearch	1.7216490	270	localsearch:localsearch	1.356667
dummy	instance	1.2096820	269	instance:dummy	1.473333
dummy	algorithm	1.4968198	268	algorithm:dummy	1.473333
dummy	localsearch	1.3455830	268	localsearch:dummy	1.473333
localsearch	algorithm	2.0849941	245	algorithm:localsearch	1.356667
localsearch	dummy	2.2400707	241	dummy:localsearch	1.356667
localsearch	instance	2.2354770	239	instance:localsearch	1.356667
algorithm	dummy	2.3485866	238	dummy:algorithm	1.400000
algorithm	localsearch	2.3540165	237	localsearch:algorithm	1.400000
algorithm	instance	2.5353357	233	instance:algorithm	1.400000

For this example we aggregate bidirectional (`param1:param2` and `param2:param2`) interactions, this is done given that we assume that the hierarchy of the interaction is not well represented in the forest and this should be analyzed separately. We also filter interactions using the reference parameter to remove all not relevant interactions. Once this process is done, the importance of most relevant interactions is summarized.

Relevant parameter interactions:

```
kable(interactions_frame) %>% kable_styling(latex_options="scale_down")
```

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth

2. Configuration normalized performance as dependent variable

The data set used to train this model defines the variable to predict as the normalized performance. Performance is normalized as:

`(performance - min_performance) / (max_performance - min_performance)`

where `performance` is the performance of a of a configuration in an instance and `min_performance` and `max_performance` are the minimum and maximum performance on an instance used during the execution of irace.

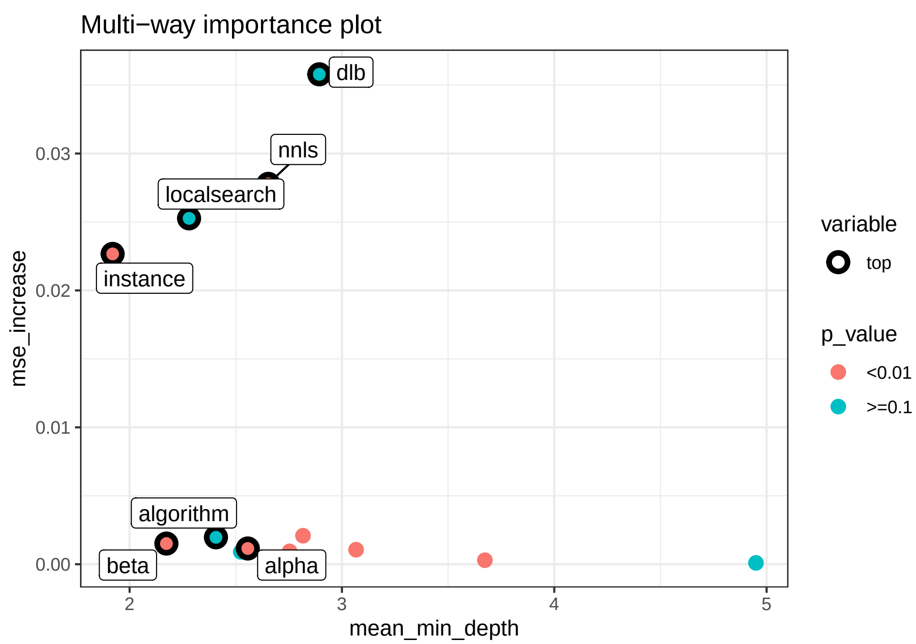
The parameter importance measures:

```
load("../model_data/model-acotsp2000-nperformance.Rdata")
importance_frame = model$importance_frame
important_parameters = model$important_parameters
full_interactions_frame = model$full_interactions_frame
interactions_frame = model$interactions_frame
kable(importance_frame[order(importance_frame[,"mean_min_depth"]),])  %>%
      kable_styling(latex_options="scale_down")
```

	variable	mean_min_depth	no_of_nodes	mse_increase	node_purity_increase	no_of_trees	times_a_root	p_value
8	instance	1.920000	73018	0.0226687	56.8832190	300	33	0.0e+00
4	beta	2.173333	48880	0.0015100	4.6549372	300	2	0.0e+00
9	localsearch	2.280000	5135	0.0252654	62.9467099	300	60	1.0e+00
1	algorithm	2.406667	7595	0.0019665	4.3988104	300	15	1.0e+00
7	elitistants	2.523333	3229	0.0009034	1.8523995	300	0	1.0e+00
2	alpha	2.556667	50032	0.0011509	4.1140106	300	0	0.0e+00
10	nnls	2.653333	38267	0.0277859	73.0291873	300	67	0.0e+00
3	ants	2.753333	45282	0.0009413	3.3455102	300	8	0.0e+00
11	q0	2.816667	31090	0.0020826	5.3741021	300	18	1.8e-06
5	dlb	2.893333	4977	0.0357884	94.8958463	300	93	1.0e+00
13	rho	3.066667	45181	0.0010601	2.9559268	300	4	0.0e+00
6	dummy	3.673333	36241	0.0002927	1.6625274	300	0	0.0e+00
12	rasrank	4.950000	5135	0.0000976	0.2391778	300	0	1.0e+00

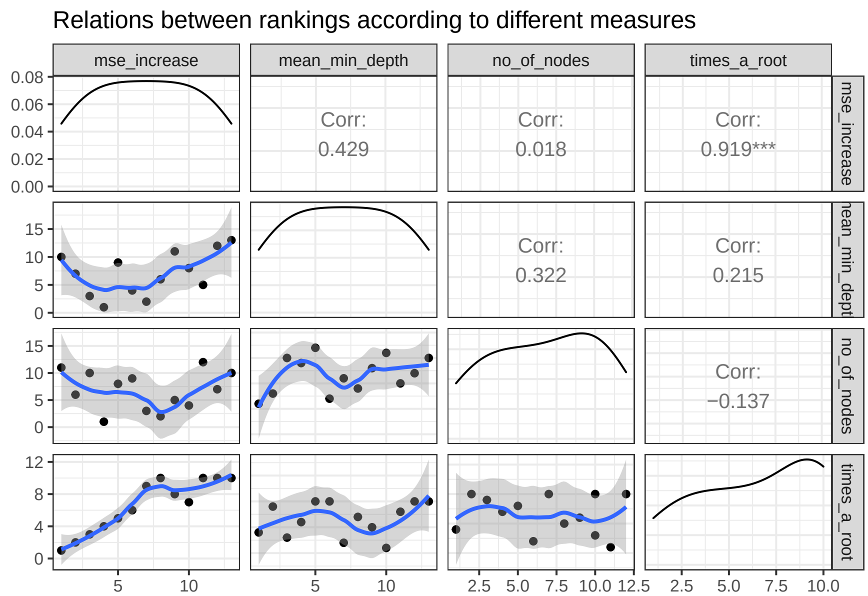
We plot importance as above to visualize how the mse increase and mean min depth are related:

```
suppressWarnings(plot_multi_way_importance(importance_frame, size_measure = "p_value",
                                           y_measure="mse_increase", x_measure="mean_min_depth",
                                           no_of_labels=7))
```



We visualize the relationship between importance measures, this could help to understand which indicator is more suited or can be used as a complement of other.

```
plot_importance_rankings(importance_frame, measures=c("mse_increase", "mean_min_depth",
                                                      "no_of_nodes", "times_a_root"))
```



Important parameters:

```r
print(important_parameters)
```

```
## [1] "instance"    "localsearch" "algorithm"   "dummy"
```

Parameter interaction importance:

```r
kable(full_interactions_frame)  %>%  kable_styling(latex_options="scale_down")
```

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth
instance	instance	0.8546099	282	instance:instance	1.313333
dummy	dummy	1.5496454	267	dummy:dummy	1.456667
instance	dummy	1.5869976	260	dummy:instance	1.313333
instance	algorithm	1.6143617	257	algorithm:instance	1.313333
dummy	instance	1.8184752	255	instance:dummy	1.456667
instance	localsearch	1.7011584	253	localsearch:instance	1.313333
algorithm	algorithm	2.2731206	244	algorithm:algorithm	1.410000
algorithm	instance	2.2062766	243	instance:algorithm	1.410000
localsearch	localsearch	2.6149173	241	localsearch:localsearch	1.393333
localsearch	instance	2.3448936	240	instance:localsearch	1.393333
dummy	localsearch	2.2261466	236	localsearch:dummy	1.456667
algorithm	localsearch	2.4487943	228	localsearch:algorithm	1.410000
algorithm	dummy	2.6534279	226	dummy:algorithm	1.410000
localsearch	dummy	2.6002364	226	dummy:localsearch	1.393333
dummy	algorithm	2.6138652	223	algorithm:dummy	1.456667
localsearch	algorithm	2.6457801	223	algorithm:localsearch	1.393333

For this example we aggregate bidirectional (`param1:param2` and `param2:param2`) interactions, this is done given that we assume that the hierarchy of the interaction is not well represented in the forest and this should be analyzed separately. We also filter interactions using the reference parameter to remove all not relevant interactions. Once this process is done, the importance of most relevant interactions is summarized.

Relevant parameter interactions:

```r
kable(interactions_frame) %>% kable_styling(latex_options="scale_down")
```

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth

3.Configuration performancequantile as dependent variable

In this example we use the performance quartile as the dependent variable when training, thus the trained random forest predicts the mean performance quartile of a configuration.

Parameter importance:

```
load("../model_data/model-acotsp2000-qperformance.Rdata")
importance_frame = model$importance_frame
full_interactions_frame = model$full_interactions_frame
interactions_frame = model$interactions_frame
important_parameters = model$important_parameters
kable(importance_frame[order(importance_frame[,"mean_min_depth"]),]) %>%
      kable_styling(latex_options="scale_down")
```

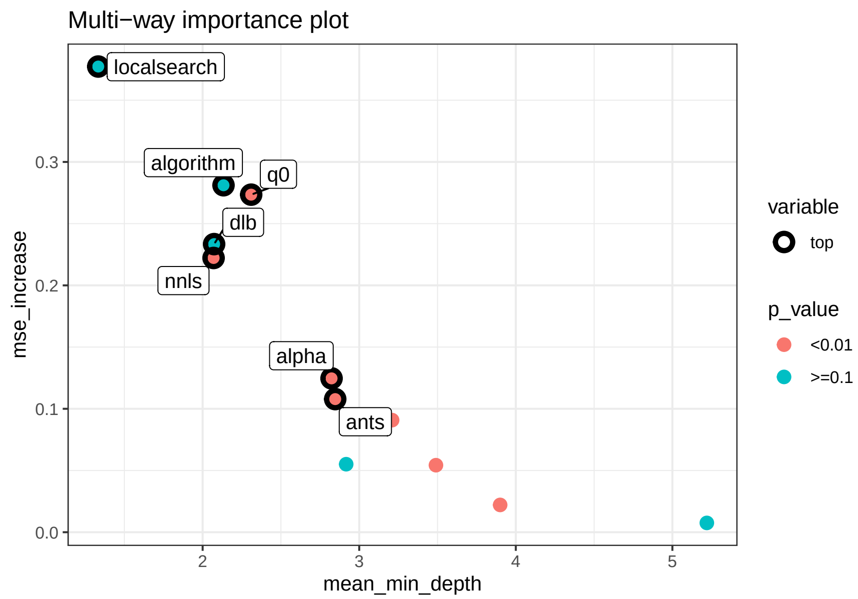
	variable	mean_min_depth	no_of_nodes	mse_increase	node_purity_increase	no_of_trees	times_a_root	p_value
9	localsearch	1.333333	4506	0.3772923	869.90625	300	71	1
10	nnls	2.070000	36805	0.2221869	509.57355	300	22	0
5	dlb	2.073333	4003	0.2334218	412.74647	300	31	1
1	algorithm	2.133333	4983	0.2811009	721.52807	300	76	1
11	q0	2.310000	28665	0.2734797	811.56088	300	67	0
8	instance	2.640000	48165	0.2613134	1032.57619	300	0	0
2	alpha	2.823333	43611	0.1246501	353.58246	300	3	0
3	ants	2.846667	40403	0.1078496	317.12365	300	15	0
7	elitistants	2.916667	2727	0.0551079	110.81351	300	9	1
13	rho	3.210000	40363	0.0908216	250.99307	300	6	0
4	beta	3.490000	42554	0.0543049	243.67337	300	0	0
6	dummy	3.900000	32586	0.0221166	128.59244	300	0	0
12	rasrank	5.220000	5109	0.0075453	20.57233	300	0	1

As expected the instance is the most important variable, the other parameters do not have the same ranking in the different benchmarks.

We plot the importance measures min mean depth and the mse increase to observe better this difference. The instance is removed in the plot so its possible to see more clearly other parameters.

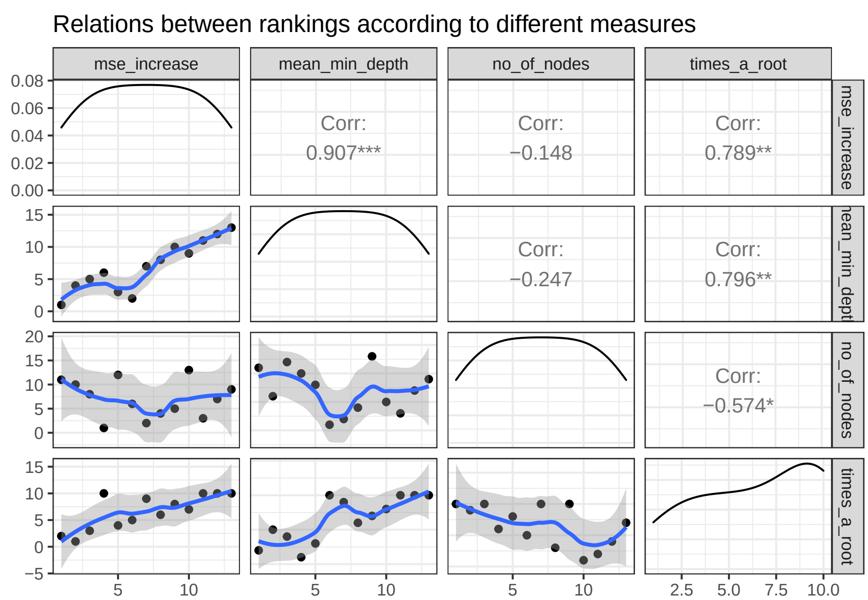
```
plot_multi_way_importance(importance_frame[importance_frame[,"variable"]!="instance",], size_measure =
                          y_measure="mse_increase", x_measure="mean_min_depth", no_of_labels=7)
```

```
## Warning: Using alpha for a discrete variable is not advised.
```



We visualize the relationship between importance measures, this could help to understand which indicator is more suited or can be used as a complement of other.

```
plot_importance_rankings(importance_frame, measures=c("mse_increase", "mean_min_depth",
                                                       "no_of_nodes", "times_a_root"))
```



Filtered important parameters:

```
print(important_parameters)
```

```
## [1] "localsearch" "algorithm"   "dummy"
```

We use the mean_min_depth measure as reference given that this measure can be extended to assess interactions and run the interaction analysis to find the importance of interactions between the selected important parameters. This is done by extending the definition of mean_min_depth to be calculated in max subtrees in which one variable is root.

```r
kable(full_interactions_frame)  %>%  kable_styling(latex_options="scale_down")
```

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth
algorithm	algorithm	0.8237548	261	algorithm:algorithm	1.114200
localsearch	localsearch	0.7476373	259	localsearch:localsearch	1.070000
dummy	algorithm	0.6514567	258	algorithm:dummy	1.104633
dummy	dummy	0.8817679	256	dummy:dummy	1.104633
dummy	localsearch	1.0909323	235	localsearch:dummy	1.104633
localsearch	algorithm	1.5761784	223	algorithm:localsearch	1.070000
algorithm	localsearch	1.7265645	214	localsearch:algorithm	1.114200
localsearch	dummy	1.7640389	211	dummy:localsearch	1.070000
algorithm	dummy	1.7557181	210	dummy:algorithm	1.114200

We aggregate interactions with their inverse given that for now we are not interested in the direction of the interaction. Once we aggregate to have bidirectional interactions and filter dummy interactions, we get a matrix where the relevant importance of interactions are summarized:

```r
kable(interactions_frame)  %>%  kable_styling(latex_options="scale_down")
```

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth