ACOTSP 1000-4500: 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 (long)
- 11 parameters
- 400 instances sizes (1000,1500,2000,2500,3000,3500,4000,4500)
- 3000 experiments configuration budget / 5000 experiments configuration budget

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.

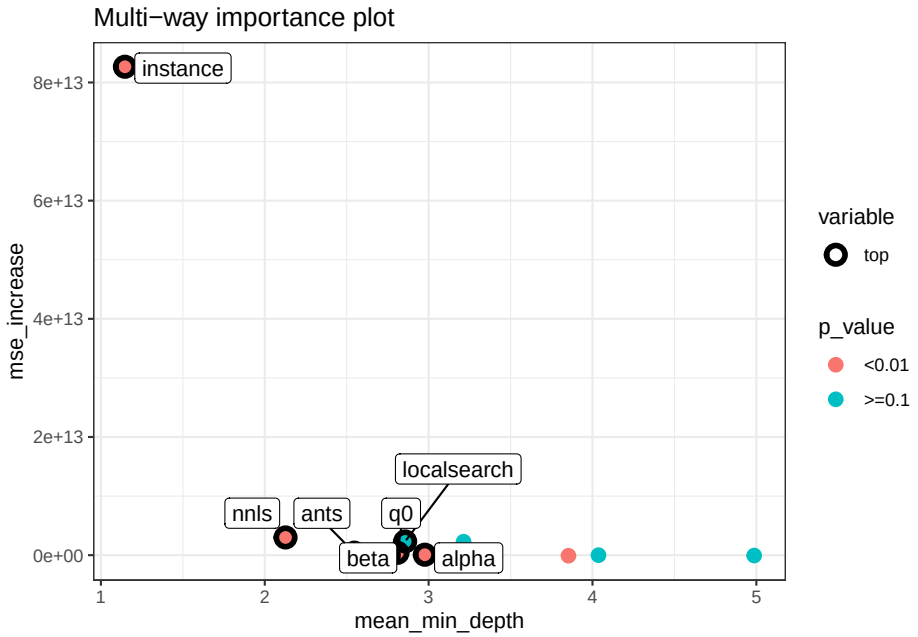
We show importance measures ordered by the mean min depth given that the interaction analysis is based on this measure.

```
load("../model_data/model-acotsp1000-4500-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
8	instance	1.146667	52322	8.266043e+13	2.130975e+17	300	84	0
10	nnls	2.126667	48190	2.996094e+12	5.761122e+15	300	50	0
3	ants	2.546667	55389	4.217218e+11	1.546904e+15	300	21	0
4	beta	2.756667	58415	3.514062e+10	1.159752e+15	300	14	0
11	q0	2.810000	42309	3.958169e+11	1.206143e+15	300	11	0
9	localsearch	2.856667	13279	2.319262e+12	4.370366e+15	300	50	1
2	alpha	2.976667	61437	7.520523e+10	1.207944e+15	300	7	0
13	rho	3.026667	55121	-1.483487e+10	1.163212e+15	300	7	0
1	algorithm	3.136667	14319	2.119131e+11	6.259429e+14	300	1	1
5	dlb	3.213333	8604	2.281706e+12	4.556949e+15	300	51	1
6	dummy	3.853333	41808	-9.520383e+10	5.136826e+14	300	2	0
7	elitistants	4.036667	6700	-3.869850e+08	2.604668e+14	300	1	1
12	rasrank	4.986667	5979	-6.150027e+10	1.612699e+14	300	1	1

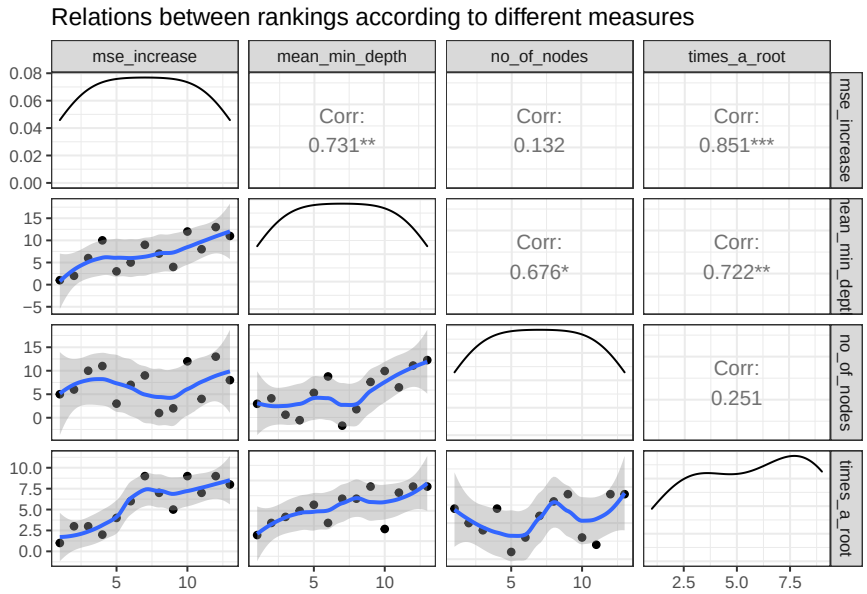
We can plot two importance measures using the randomForestExplainer package, we choose the to show the mean min depth in the x axis and the mse increase in the y axis. Top 7 variables are highlighted. In this case, since we are not predicting performance the effect of instance variable in model performance 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).

```
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] "instance"    "nnls"        "ants"        "localsearch" "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	nnls	1.158784	296	nnls:instance	1.620000
nnls	nnls	1.475011	295	nnls:nnls	1.710000
ants	localsearch	1.228986	294	localsearch:ants	1.623333
instance	localsearch	1.178311	294	localsearch:instance	1.620000
ants	nnls	1.154764	293	nnls:ants	1.623333
localsearch	localsearch	1.829966	293	localsearch:localsearch	1.710000
nnls	localsearch	1.498885	293	localsearch:nnls	1.710000
ants	ants	1.344279	289	ants:ants	1.623333
dummy	nnls	1.543333	288	nnls:dummy	1.556667
instance	ants	1.357748	288	ants:instance	1.620000
ants	dummy	1.512838	284	dummy:ants	1.623333
dummy	dummy	1.533108	284	dummy:dummy	1.556667
dummy	localsearch	1.691486	284	localsearch:dummy	1.556667
localsearch	nnls	1.919730	284	nnls:localsearch	1.710000
ants	instance	1.570203	282	instance:ants	1.623333
instance	dummy	1.664752	282	dummy:instance	1.620000
dummy	instance	1.582939	281	instance:dummy	1.556667
instance	instance	1.640372	281	instance:instance	1.620000
dummy	ants	1.691847	280	ants:dummy	1.556667
nnls	instance	2.585068	262	instance:nnls	1.710000
nnls	ants	2.734775	258	ants:nnls	1.710000
nnls	dummy	2.830462	255	dummy:nnls	1.710000
localsearch	ants	3.399820	247	ants:localsearch	1.710000
localsearch	dummy	3.373705	243	dummy:localsearch	1.710000
localsearch	instance	3.421655	243	instance:localsearch	1.710000

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
ants	localsearch	1.228986	541	localsearch:ants	1.623333
nnls	localsearch	1.498885	577	localsearch:nnls	1.710000

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.

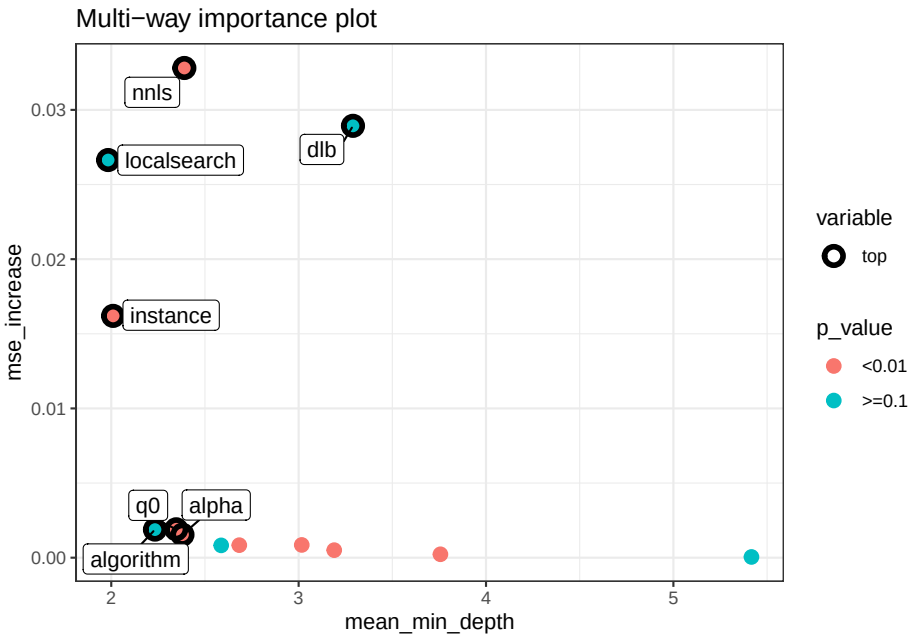
The parameter importance measures:

```
load("../model_data/model-acotsp1000-4500-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
9	localsearch	1.983333	5020	0.0266410	69.396473	300	65	1
8	instance	2.010000	73810	0.0161997	44.522499	300	10	0
1	algorithm	2.233333	6406	0.0018821	4.245921	300	17	1
11	q0	2.346667	33405	0.0019020	6.880823	300	35	0
2	alpha	2.380000	49682	0.0015422	4.897223	300	12	0
10	nnls	2.390000	39464	0.0328080	90.270016	300	86	0
7	elitistants	2.586667	3730	0.0008248	1.796536	300	3	1
4	beta	2.683333	48874	0.0008389	2.873481	300	0	0
3	ants	3.016667	45773	0.0008536	2.354539	300	0	0
13	rho	3.190000	45984	0.0005050	2.303628	300	0	0
5	dlb	3.290000	4933	0.0289325	79.842032	300	72	1
6	dummy	3.756667	36030	0.0002280	1.234274	300	0	0
12	rasrank	5.416667	4624	0.0000489	0.151571	300	0	1

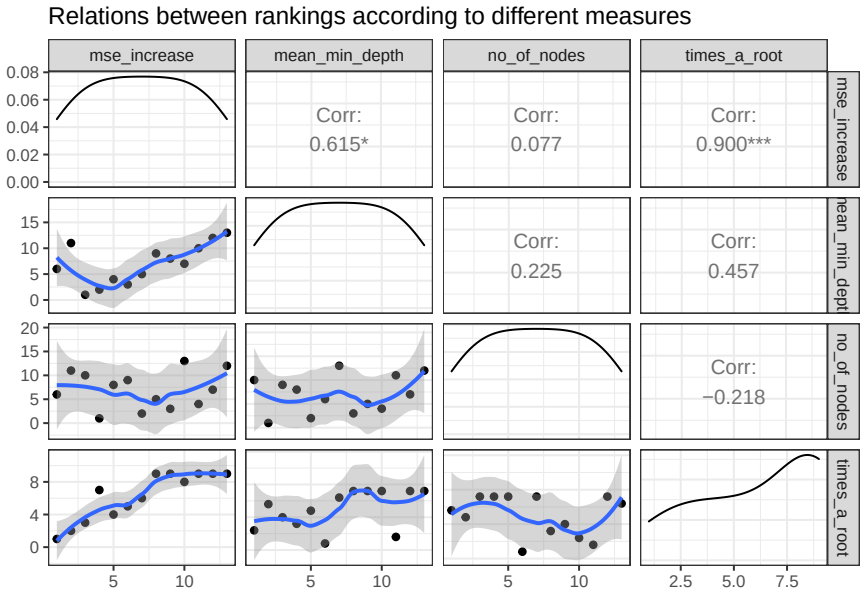
We plot importance as above to visualize how the mse increase and mean min depth are related:

```
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:

```
print(important_parameters)
```

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

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	localsearch	0.7678571	280	localsearch:instance	1.396667
algorithm	algorithm	1.4161786	277	algorithm:algorithm	1.513333
dummy	localsearch	0.9510357	277	localsearch:dummy	1.386667
instance	algorithm	0.9572857	276	algorithm:instance	1.396667
dummy	algorithm	1.0019643	275	algorithm:dummy	1.386667
instance	dummy	0.9862024	273	dummy:instance	1.396667
instance	instance	1.1469048	272	instance:instance	1.396667
dummy	instance	1.2116071	265	instance:dummy	1.386667
localsearch	localsearch	1.9269524	264	localsearch:localsearch	1.500000
dummy	dummy	1.2889405	263	dummy:dummy	1.386667
localsearch	algorithm	1.9743929	247	algorithm:localsearch	1.500000
algorithm	instance	2.9568452	225	instance:algorithm	1.513333
localsearch	instance	2.8818452	225	instance:localsearch	1.500000
localsearch	dummy	2.5574762	224	dummy:localsearch	1.500000
algorithm	localsearch	2.8161071	223	localsearch:algorithm	1.513333
algorithm	dummy	3.0696071	211	dummy:algorithm	1.513333

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

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-acotsp1000-4500-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")
```

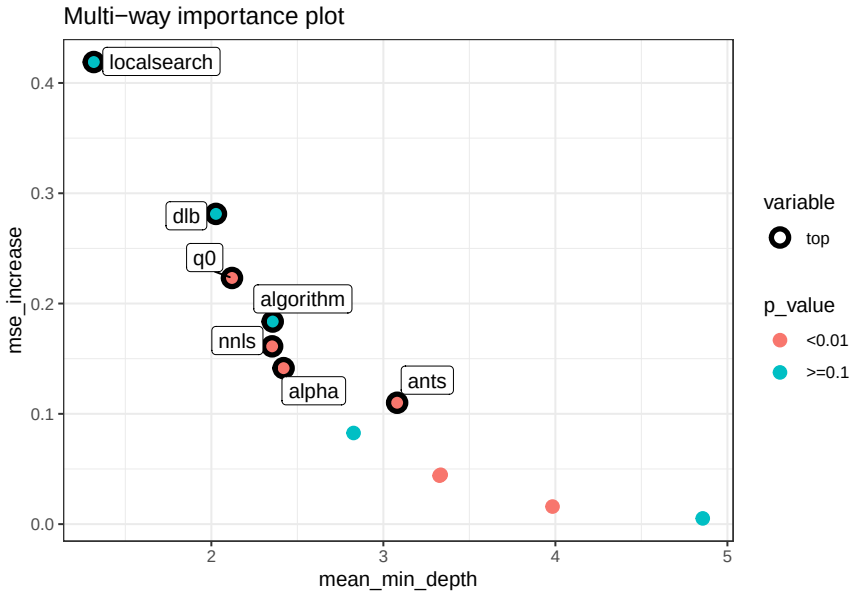
	variable	mean_min_depth	no_of_nodes	mse_increase	node_purity_increase	no_of_trees	times_a_root	p_value
9	localsearch	1.316667	3829	0.4189815	1108.10341	300	80	1
5	dlb	2.026667	3372	0.2812971	523.94304	300	41	1
11	q0	2.120000	32227	0.2231810	879.17507	300	80	0
10	nnls	2.353333	38042	0.1612048	447.75094	300	18	0
1	algorithm	2.356667	4565	0.1836905	478.95533	300	43	1
2	alpha	2.420000	44550	0.1414235	406.76871	300	16	0
8	instance	2.616667	51678	0.2217258	901.75882	300	0	0
7	elitistants	2.826667	2209	0.0826420	179.87060	300	14	1
3	ants	3.080000	41031	0.1100577	302.25395	300	4	0
4	beta	3.326667	43323	0.0439569	215.24804	300	2	0
13	rho	3.333333	41227	0.0447672	204.24158	300	2	0
6	dummy	3.983333	32840	0.0159335	115.85962	300	0	0
12	rasrank	4.856667	4579	0.0051905	23.78457	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.

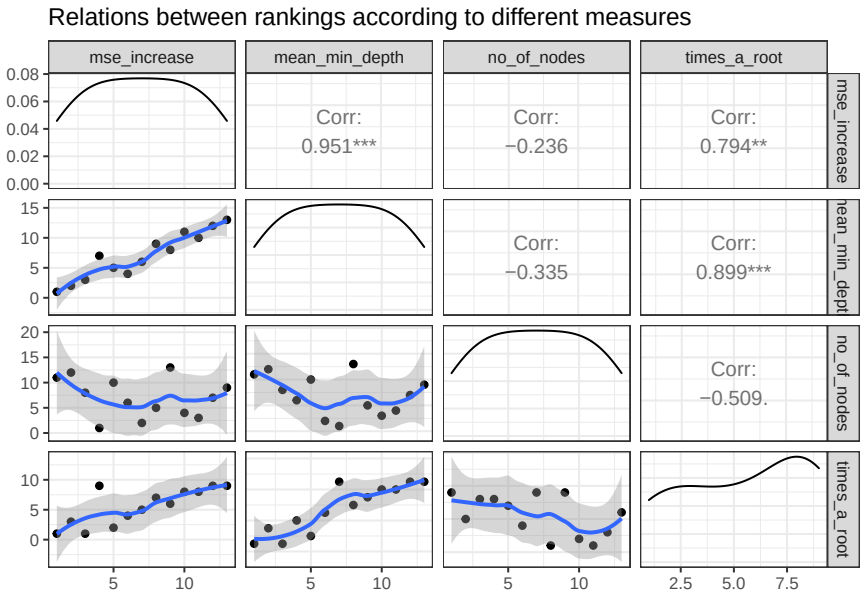
```
plot_multi_way_importance(importance_frame[importance_frame[,"variable"]!="instance",], size_measure =
                          y_measure="mse_increase", x_measure="mean_min_depth", no_of_labels=7)
```

```
## 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.

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

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth
dummy	dummy	0.6678967	271	dummy:dummy	1.147078
algorithm	algorithm	0.9712177	263	algorithm:algorithm	0.980000
dummy	algorithm	1.0658672	252	algorithm:dummy	1.147078
localsearch	localsearch	1.3901599	248	localsearch:localsearch	1.116667
dummy	localsearch	1.2174662	239	localsearch:dummy	1.147078
localsearch	algorithm	1.6363469	228	algorithm:localsearch	1.116667
localsearch	dummy	1.8297770	219	dummy:localsearch	1.116667
algorithm	dummy	1.8230263	218	dummy:algorithm	0.980000
algorithm	localsearch	1.7202952	217	localsearch:algorithm	0.980000

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:

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

variable	root_variable	mean_min_depth	occurrences	interaction	uncond_mean_min_depth