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

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 normalized ranking
2. configuration imputed ranking
3. configuration imputed ranking 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 we use the *ranking as the dependent variable* when training, thus the trained random forest predicts the mean ranking of a configuration. In these experiments the variable instance should not be a good as predictor compared to a model trained to predict performance directly. Despite this, interactions of the instance variable with other variables are possible and might indicate heterogeneity of the benchmark. In this model all irace data execution was used for training. Note that due to the focused nature of irace data (given model convergence) there might be contradicting or not consistent interactions in the data.

1. Configuration normalized ranking as dependent variable

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

`(rank - 1) / (n_instances - 1)`

where `rank` is the rank of a of a configuration in an instance and `n_instances` is the number of instances used during the execution of irace.

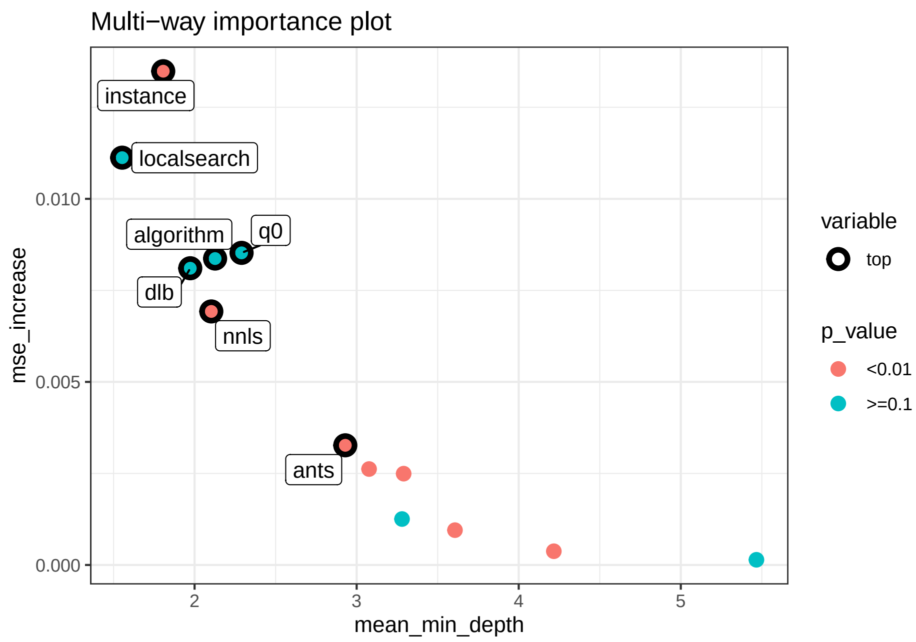
We show importance measures ordered by the mean min depth given that the interaction analysis is based on this measure.

```
load("../model_data/model-acotsp2000-ranking.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	1.553333	6522	0.0111240	39.4276270	300	63	1
8	instance	1.806667	73126	0.0134860	34.6621635	300	19	0
5	dlb	1.973333	4917	0.0081042	21.6919222	300	29	1
10	nnls	2.103333	39862	0.0069245	19.4169652	300	26	0
1	algorithm	2.126667	9817	0.0083689	36.8292841	300	69	1
11	q0	2.290000	30991	0.0085246	37.9757083	300	73	1
3	ants	2.930000	48591	0.0032685	8.8095081	300	10	0
2	alpha	3.076667	53043	0.0026209	6.9686851	300	1	0
7	elitistants	3.280000	4673	0.0012551	3.2948551	300	2	1
13	rho	3.290000	48451	0.0024940	5.5629780	300	8	0
4	beta	3.606667	51717	0.0009512	3.6010366	300	0	0
6	dummy	4.216667	38495	0.0003754	1.7795626	300	0	0
12	rasrank	5.466667	6631	0.0001417	0.5524759	300	0	1

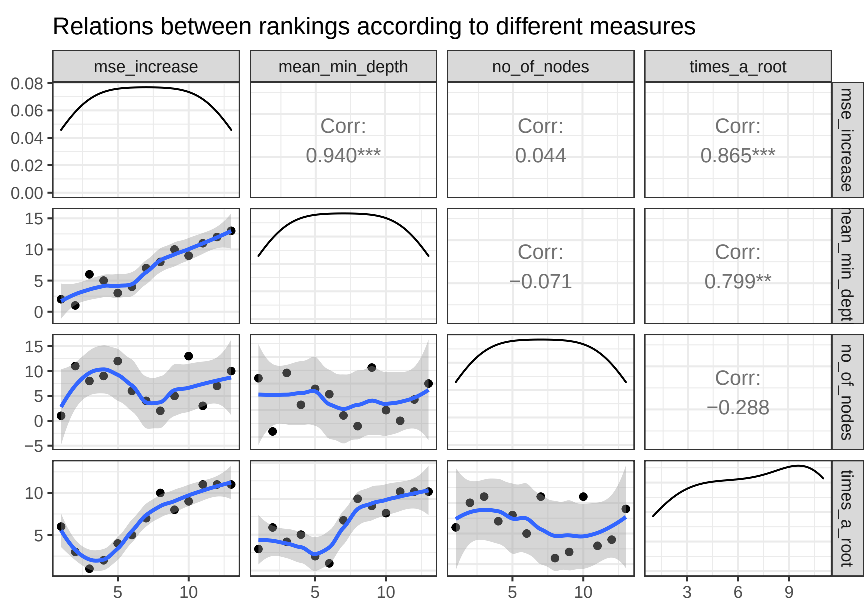
We can plot two importance measures using the randomForestExplainer package, we choose the to show the mean min depth in the x axis and the mse increase in the y axis. Top 7 variables are highlighted. In this case, since we are not predicting performance the effect of instance variable in model performance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

```
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" "instance"    "algorithm"    "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	0.7253521	284	instance:instance	1.423333
instance	dummy	1.1312911	277	dummy:instance	1.423333
dummy	dummy	1.4176526	276	dummy:dummy	1.193333
instance	algorithm	1.1646127	273	algorithm:instance	1.423333
instance	localsearch	1.2794836	268	localsearch:instance	1.423333
localsearch	localsearch	1.8471244	259	localsearch:localsearch	1.370000
algorithm	algorithm	2.0070423	256	algorithm:algorithm	1.553333
algorithm	dummy	2.0143897	247	dummy:algorithm	1.553333
dummy	algorithm	1.9867958	243	algorithm:dummy	1.193333
dummy	instance	2.1362441	240	instance:dummy	1.193333
algorithm	instance	2.2109390	238	instance:algorithm	1.553333
dummy	localsearch	2.2506690	233	localsearch:dummy	1.193333
algorithm	localsearch	2.4092019	232	localsearch:algorithm	1.553333
localsearch	dummy	2.5668545	232	dummy:localsearch	1.370000
localsearch	instance	2.4449531	229	instance:localsearch	1.370000
localsearch	algorithm	2.6408451	224	algorithm:localsearch	1.370000

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 imputed ranking as dependent variable

For training the model we must impute the dependent variable (ranking) since some configurations are not executed in some instances. We assume these configurations to be worst than the configurations executed in the instance, after imputation all configurations have a ranking assigned for each instance.

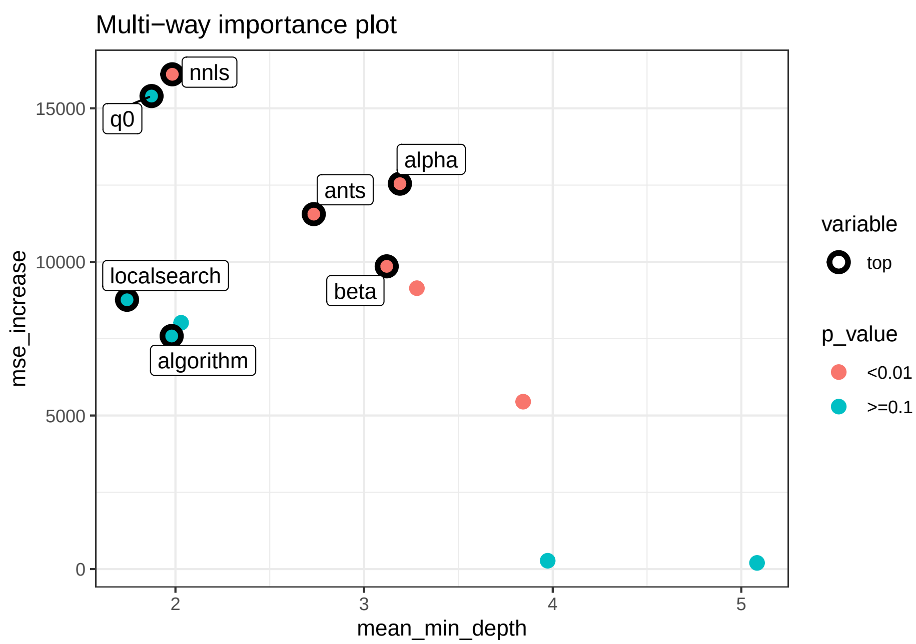
The parameter importance measures:

```
load("../model_data/model-acotsp2000-iranking.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	1.743333	12241	8766.5496	22040391	300	47	1
11	q0	1.873333	44743	15399.9337	61101158	300	90	1
1	algorithm	1.980000	17418	7587.9461	26834053	300	71	1
10	nnls	1.983333	65884	16107.9869	50527537	300	23	0
5	dlb	2.030000	9098	8018.8966	18792288	300	35	1
8	instance	2.173333	228173	6835.5666	85431332	300	0	0
3	ants	2.733333	84221	11555.6488	39275373	300	20	0
4	beta	3.120000	88219	9856.0290	39844526	300	0	0
2	alpha	3.190000	87984	12547.6868	44227778	300	1	0
13	rho	3.280000	83431	9144.9087	32223338	300	8	0
6	dummy	3.843333	66906	5449.3942	20021910	300	0	0
7	elitistants	3.973333	10786	271.4172	1318174	300	4	1
12	rasrank	5.083333	16041	200.9920	2053455	300	1	1

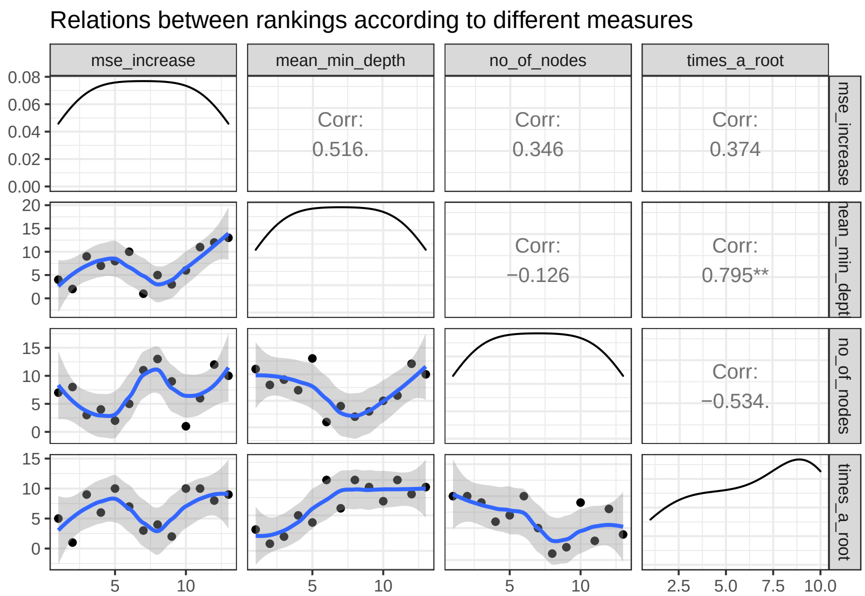
We plot importance as above to visualize how the mse increase and mean min depth are related:

```
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:

```r
print(important_parameters)
```

```
## [1] "localsearch" "algorithm"   "nnls"        "ants"        "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
ants	nnls	1.128472	288	nnls:ants	1.616667
ants	algorithm	1.049907	286	algorithm:ants	1.616667
dummy	dummy	1.209317	283	dummy:dummy	1.640000
ants	dummy	1.230347	282	dummy:ants	1.616667
nnls	dummy	1.157431	282	dummy:nnls	1.606667
ants	localsearch	1.179630	281	localsearch:ants	1.616667
nnls	nnls	1.323981	281	nnls:nnls	1.606667
nnls	ants	1.241852	280	ants:nnls	1.606667
ants	ants	1.331979	279	ants:ants	1.616667
dummy	nnls	1.183565	278	nnls:dummy	1.640000
nnls	algorithm	1.482176	278	algorithm:nnls	1.606667
nnls	localsearch	1.487269	278	localsearch:nnls	1.606667
dummy	algorithm	1.368889	276	algorithm:dummy	1.640000
dummy	ants	1.328056	276	ants:dummy	1.640000
algorithm	algorithm	2.131898	272	algorithm:algorithm	1.610000
dummy	localsearch	1.481019	272	localsearch:dummy	1.640000
localsearch	localsearch	2.158565	268	localsearch:localsearch	1.503333
localsearch	algorithm	2.470370	248	algorithm:localsearch	1.503333
algorithm	nnls	2.450185	247	nnls:algorithm	1.610000
algorithm	localsearch	2.745139	240	localsearch:algorithm	1.610000
localsearch	nnls	2.858796	238	nnls:localsearch	1.503333
localsearch	dummy	2.864201	237	dummy:localsearch	1.503333
algorithm	dummy	2.961620	236	dummy:algorithm	1.610000
algorithm	ants	2.911863	233	ants:algorithm	1.610000
localsearch	ants	3.108877	227	ants:localsearch	1.503333

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	nnls	1.128472	568	nnls:ants	1.616667

3.Configuration imputed ranking quantile as dependent variable

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

In this example 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).

Parameter importance:

```
load("../model_data/model-acotsp2000-qranking.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")
```

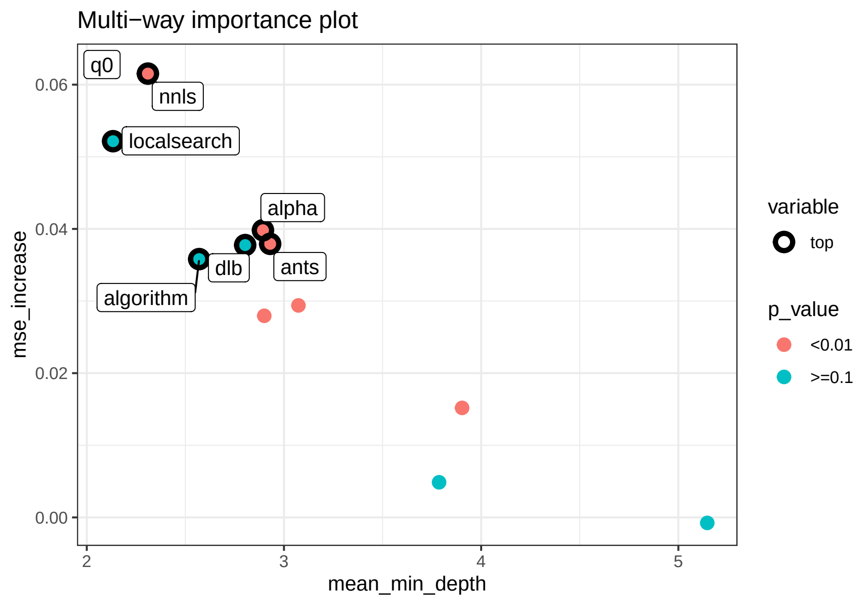
	variable	mean_min_depth	no_of_nodes	mse_increase	node_purity_increase	no_of_trees	times_a_root	p_value
8	instance	1.156667	76600	0.6941809	5404.76736	300	84	0.0000000
11	q0	2.103333	41294	0.0625137	406.29527	300	75	0.9976283
9	localsearch	2.133333	8842	0.0521620	230.25365	300	34	1.0000000
10	nnls	2.310000	55801	0.0615319	379.73586	300	16	0.0000000
1	algorithm	2.570000	10725	0.0358129	195.65640	300	52	1.0000000
5	dlb	2.803333	8653	0.0377590	136.56240	300	19	1.0000000
2	alpha	2.893333	69841	0.0398152	353.08710	300	0	0.0000000
4	beta	2.900000	69680	0.0279522	299.38854	300	0	0.0000000
3	ants	2.930000	65755	0.0379270	288.15563	300	13	0.0000000
13	rho	3.073333	65500	0.0293923	265.24403	300	3	0.0000000
7	elitistants	3.786667	6810	0.0048689	27.26944	300	4	1.0000000
6	dummy	3.903333	53163	0.0151822	171.59238	300	0	0.0000000
12	rasrank	5.146667	11354	-0.0007622	16.96926	300	0	1.0000000

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.

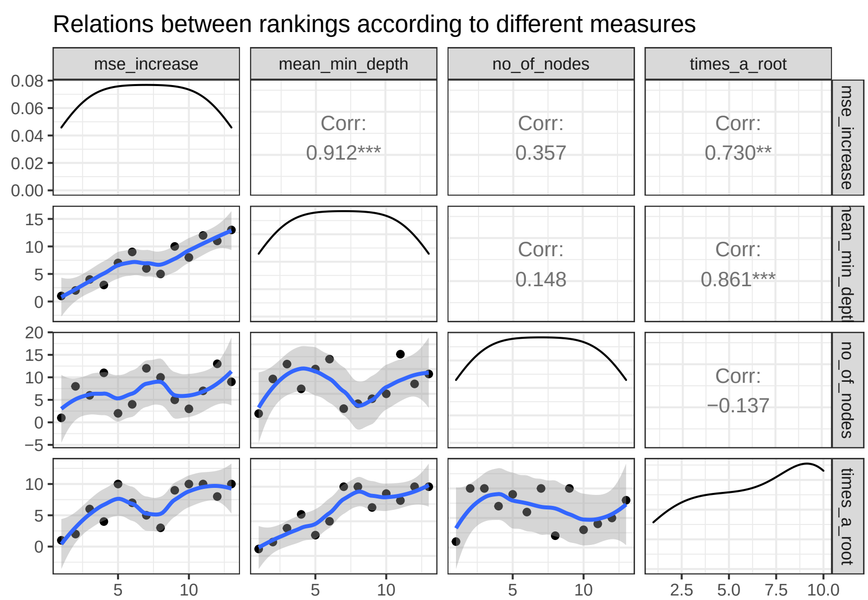
```
plot_multi_way_importance(importance_frame[importance_frame[,"variable"]!="instance",], size_measure =
                          y_measure="mse_increase", x_measure="mean_min_depth", no_of_labels=7)
```

```
## 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] "instance"    "localsearch" "nnls"        "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
instance	instance	0.9822064	281	instance:instance	1.580000
instance	algorithm	1.1695018	274	algorithm:instance	1.580000
dummy	instance	1.3916963	271	instance:dummy	1.636667
nnls	instance	1.3881376	271	instance:nnls	1.730000
instance	localsearch	1.4107948	267	localsearch:instance	1.580000
nnls	localsearch	1.5744958	267	localsearch:nnls	1.730000
instance	nnls	1.4624911	265	nnls:instance	1.580000
nnls	nnls	1.5650178	264	nnls:nnls	1.730000
dummy	localsearch	1.6110320	263	localsearch:dummy	1.636667
instance	dummy	1.4802135	263	dummy:instance	1.580000
dummy	algorithm	1.7249466	262	algorithm:dummy	1.636667
nnls	algorithm	1.7964413	261	algorithm:nnls	1.730000
dummy	dummy	1.8004626	260	dummy:dummy	1.636667
dummy	nnls	1.8520996	257	nnls:dummy	1.636667
algorithm	algorithm	2.4188612	251	algorithm:algorithm	1.613333
nnls	dummy	2.0980783	249	dummy:nnls	1.730000
algorithm	instance	2.5852669	242	instance:algorithm	1.613333
localsearch	instance	2.7497746	238	instance:localsearch	1.566667
localsearch	localsearch	3.0754448	233	localsearch:localsearch	1.566667
algorithm	localsearch	2.7903915	227	localsearch:algorithm	1.613333
localsearch	nnls	3.0346619	227	nnls:localsearch	1.566667
localsearch	algorithm	2.9713523	226	algorithm:localsearch	1.566667
algorithm	dummy	2.9814947	221	dummy:algorithm	1.613333
algorithm	nnls	3.0590747	221	nnls:algorithm	1.613333
localsearch	dummy	3.3514591	209	dummy:localsearch	1.566667

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