Week 8

- Changed optimization, evaluation metric and testing set, and new switching points
- For the optimization, I used asf's already existing configuration space, as suggested by Hadar
- Evaluation metric:
  In our case, m_s is
  $$\hat{m}_s = \frac{m_s - m_{VBS}}{m_{SBS} - m_{VBS}}$$
  the sum of precisions of our selector across the test set
- Test set are now 10 runs on each instance, so:
  - We train on runs 0-19, test on runs 20-29
- We now consider the following switching points: Up to 96 evaluations, every 8 evaluations (as 8 is the population size of CMA-ES), then 100, 150, 200,...,1000 (where 1000 means CMA-ES runs to the end, i.e. we never switch)

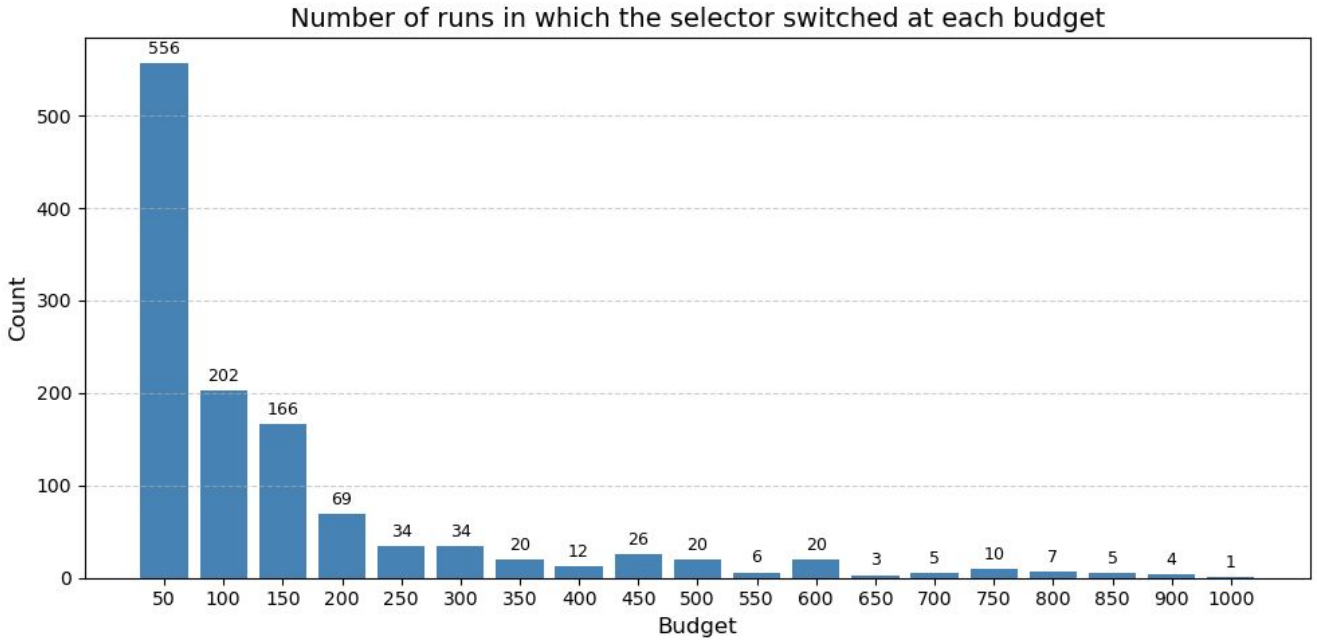
Results with switching points every 50 evaluations

- In the new metric, a value greater than 1 means the selector is worse than the SBS and we want to get as close to 0 as possible
- SBS: Switching to non-elitist at budget 50
- The selector and budget 50 are basically the same

Method	Ratio
static_B50	0.8046751434116026
selector_precision	0.8077026824431898
static_B200	0.8586391475049477
static_B100	0.8873436230701948
static_B150	0.9074152133760168
static_B350	0.9091574385324336
static_B450	0.9113697716564713
static_B300	0.9137920329685639
static_B400	0.9266049225310287
static_B250	0.9338964254691053
static_B500	0.947583361910235
static_B550	0.9481771419072482
static_B600	0.9513590665512437
static_B650	0.9565055112782784
static_B700	1.0621030408897698
static_B750	1.283033425284697
static_B800	1.418599310629438
static_B850	1.7669272606205617
static_B900	2.121320399943321
static_B950	2.828002552258648
static_B1000	4.275246125578036

Results with switching points every 50 evaluations

How often did we switch at which switching point across all runs?



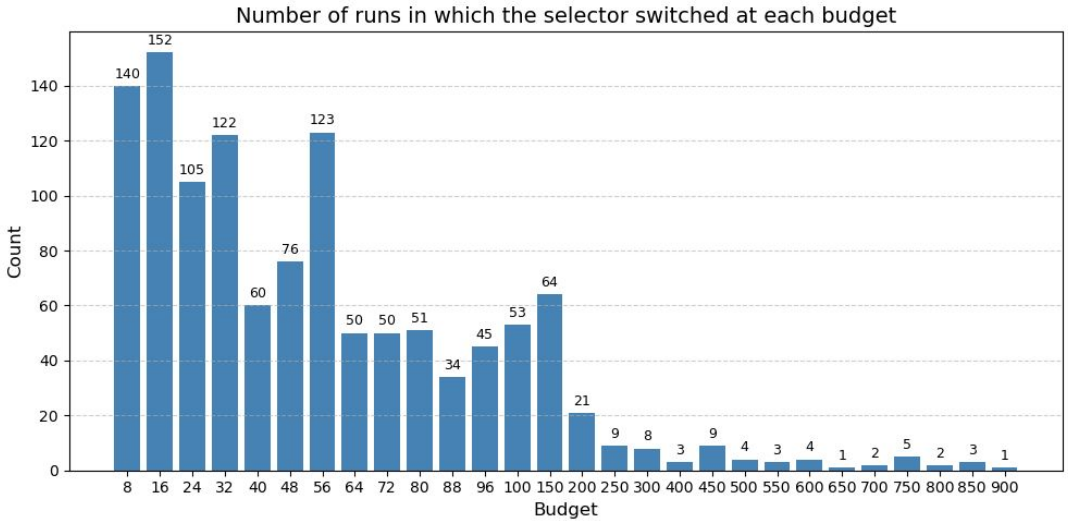
Results with new switching points

- I now consider the following switching points:
  Multiples of 8 until 96, starting from 100 multiples of 50 until 1000.
- SBS here: Non-elitist at budget 16
- The selector here is worse than the one that only switches every 50 evals!
  Probably it gets favors early switching points too much, although they are quite bad (budgets 8, 16)

Method	Ratio
static_B80	0.8343698876143243
static_B56	0.8431062571644096
static_B64	0.865747418437369
static_B24	0.8695766953339663
static_B40	0.874061958394088
static_B96	0.8749348244632318
selector_precision	0.8819141752446471
static_B200	0.8864525814801185
static_B100	0.9095093121298923
static_B48	0.9111641348804176
static_B150	0.9256317196040164
static_B350	0.9270311535031466
static_B450	0.9288081993450942
static_B32	0.9297976509890551
static_B72	0.9301458725592493
static_B300	0.9307538689849849
static_B400	0.9410457603249096
static_B250	0.9469026246627765
static_B88	0.9556409676800052
static_B500	0.9578965899148633
static_B550	0.9583735408149151
static_B600	0.9609294063383146
static_B650	0.9650632630815067
static_B16	0.9933172712163364
static_B700	1.049883966110878
static_B750	1.227345224853125
static_B800	1.3362378641416175
static_B8	1.3913457921804038
static_B850	1.6160305989880546
static_B900	1.9006951677212154
static_B950	2.4683341759274477
static_B1000	3.63082555044471

Results with new switching points

- We switch at 8 and 16 really often, although they are quite bad



Questions / Next Steps

- The current selector switches really early. This means it is really close to the static selector at budget 50 (if we switch every 50 evals) or its actually worse than some static selectors because it switches for really low budgets (8 and 16) at which we cannot choose the best second algorithm
- I think one problem currently is that we switch based on the performance of the best algorithm at each budget. This algorithm doesn’t have to be the one that is actually selected!

Selector performance prediction

- 1. Optimize static selectors on first five instances
- 2. Record their performance on these instances using five-fold cv (trained on 4, prediction on the remaining one).
- Currently, the switching prediction models predict the precision of the best algorithm for each budget. We can now train them to predict the precision of the algorithm that is actually chosen for each budget
- Another possibility would be function-specific switching: For each function, record the best static switching budget and define that budget as the best switching point for all runs on that function
- For a new run, we then predict that switching point. Because switching-points are now function-specific, we basically predict the fid for each run. We know that we can do that quite accurately. I think this could lead to improvements.
- If we try to predict selector performance, we might need more runs for both training and testing to really capture selector performance