**Completed**

* Significant change: get rid of keeping track of the first throttled fast process (or first throttled slow process). We will keep track of the sFFP as well as rank 1 of the SPs, but rank 1 might be unthrottled. With our current algorithm, there is no need for implicit indexing, which is harder to ensure being written correctly.

This has been refactored to explicitly skip any ranks whose uEFs are within the FRP threshold (FFPs and SPs) or below the floor (FQPs).

**Need Feedback/Review**

* For the paired ranking (based on earlier guidance) you are not adding reverse processes that have a rate of 0. In the current algorithm, even those should be added to the ranked\_uEF\_list. A process with an event frequency of zero can still be throttled. [we don’t know if the event frequency will be nonzero in the next snapshot, so we still need to change the rate constant for it in our updated algorithm]. This is around line 250 in the code I am looking at, and is beneath # Add to temporary master list
* if ranking\_scheme == 'paired':

This is actually OK. The unranked uEF\_list has code (see get\_process\_info, about line 155) to reset the negligible process EFs to both be None instead of the uEFs (each process pair starts with its uEFs) based on a flag (throttle\_process\_pairs), and they are both reset to None only if both are less than the NSP threshold. And we still need the code around line 250 in case we have an explicitly irreversible process.

What does this mean? I am not sure I understand what this code is for, but ranks have to be sorted by the Max\_EF in that rank.  
 # Reverse the list of FFPs to be in proper ascending order. This sorting  
 # will break ties by putting forward reactions first.  
 ascend\_FFP\_list.sort(key=lambda x: (x[1], x[2]))

This code is OK. Each of the sublists in ascend\_FFP\_list is (0) the reaction number, (1) the rate associated with the process group (in this case a pair of processes), (2) the process name, and (3) the direction of maximum rate. The rate associated with the pair of processes is the max of the forward and reverse processes. I have updated the comment to better explain this.

* SRP\_EF = ranked\_uEF\_list[BP\_index][0][1] 🡨 I don’t think that we use this anymore. Was this used for creating EF\_range in the past? I think that we would use the slowest speedrank ptEF at this point.
  + This variable and the loop that is used to populate it should be deleted since it is no longer useful.
  + The function create\_FFP\_SP\_lists should no longer return this variable, and somewhere around line 530 that means the variable SRP\_index should be removed from there also.

The purpose of SRP\_EF is to find the point separating slow and negligible processes. This point is marked with SRP\_index. This may be useful information, although we never use it in the code.

* Why is this line here: SRP\_index = BP\_index

This is related to finding the separation between slow and negligible processes (see previous point). In this case, the BP is the slowest process before negligible processes are found (e.g., the slowest FQP/FFP and no slow processes or a single FRP).

* Near line 414 you have:  
   if BP\_type is None:

# Only happens if we only have FQPs since the presence of any FFP

* + I think that this should be made into an “else” statement where “if EF\_max >= tg.FFP\_floor:” occurs. If you see no objection, then change it.

This is almost right but not quite. The issue is if we have both FFPs and FQPs, then the sFFP should be the BP, but when looping over the processes we may find a non-QE process that we don’t know about yet. We have to keep scanning processes until either (1) we find the FRP or (2) we run out of processes. Only in the first case can we make an early definitive assignment of the type of BP. In the second case, since we can’t quit, if we blindly keep assigning the BP\_type after getting fast the sFFP, we might forget that we have FFPs and wrongly say that the BP is an FQP, when it really is an FFP. The way to address this is to only assign FFP as the type to FFPs while scanning the list and not assign FQP until all other possibilities have been eliminated. This is why this check happens last, after both the FRP and FFP possibilities have been ruled out.

\*\*\*

As far as I can tell, this loop:

“for i in range(BP\_index, SRP\_index + 1):”

should simply be from range 0 to len(ranked\_uEF\_list)+1

\*\*\*

This is true. However, I chose to have temporary variables as I thought they were more descriptive. In particular, I thought that len(ranked\_uEF\_list) was not obvious enough. I have kept SRP\_index + 1 but replaced BP\_index with 0.

* Why is it useful to have “set\_compression\_scheme” tied to the SP\_ranks and FFP\_ranks? Why not just allow SPs to be throttled as well? If there are no SPs, the function that tries to throttle them will be very quick. Or am I mistaken? Why is it more efficient to have these distinctions than to not have them? Because if a list length is zero, you’re not really saving any efficiency to avoid iterating over that list.

for the compression schemes, why can’t we just have unthrottled plus the general case loop?

else: # General case -- both FFPs and SPs

for i in range(0, 4):  
 for j in range(1, 4):  
 compression\_scheme.append((j, i))

These two are related. The issue is not the empty lists *per se*. It is the fact that we will have to iterate over the non-empty lists more than we would otherwise have to. For example, if we only have slow processes, we will try to throttle the non-existent fast processes three times before attempting to throttle the slow processes at the next scale. Essentially this means we would calculate the same set of slow process throttling factors three times before being able to try the next slow throttling scale. We have the same issue with only fast processes as well. Although we iterate over the fast scales first, if the target compression is not met, we will iterate over the fast scales three times before deciding that there really aren’t any more options.

* ADD TO YOUR TO-DO LIST: Go over structures with Ashi of speedranked uEFs etc. to make sure they are still done in a way that he can add loops later.

The ranked\_uEF\_list structure is based around process groups (currently pairs). Each process group is a list of elementary process lists. Each elementary process list has the process number, the process EF, and a 2-tuple with indices into the unranked uEF\_list. The ranked\_uEF\_list is sorted according to the first process EF in each process group list. In paired ranking, the process groups are simply the forward and reverse processes, with the faster of the two being the lead process in the process group list (so that ranke\_uEF\_list is ranked by the faster EF in the pair).

I am not sure what motivated you to make “old\_uEF = old\_ptEF” but I don’t see a need for this, please remove it unless you can provide a compelling reason. We just need uEF (which is the old uEF sortof by definition).

This whole loop is fairly strange in notation: for i in xrange(fSP\_index, SP\_ranks):

fSP\_index 🡨 why not call this FRP index? Am I misunderstanding what this is? I think we should just remove it, and remove the loop that assigns it. I guess this is the “first throttled slowest process” that we don’t need to keep track of?

These are related. The loop starts at the first throttled slow process and runs to the total number of slow processes (thereby skipping the unthrottled slow processes). To bootstrap the calculation of the ATFs for the slower processes, I need the uEF and ptEF of the last process. Because this process was not throttled, they are (by definition) the same. The fSP index is **not** the FRP index (which is always trivially zero). It is the index of the first process that should be throttled, based on the FRP/Nsites threshold. I don’t keep track of the first throttled slow process beyond its immediate use in this function. In looking at the code again, it does not do much, so I have combined the two loops.

if I am not mistaken, currently you have some “Case 1” “Case 2” type implicit logic that is not necessary. Sometimes that is necessary, other times it is an attempt to create more code efficiency. For those that can be removed (i.e., they wouldn’t improve efficiency by an order of magnitude anyway during a typical snapshot), remove them for the following reasons:

1. Such implicit logic is more likely to create bugs when you are programming
2. it makes it harder for somebody else to check your code or debug
3. if somebody does not realize there is implicit logic or if they do not understand the implicit logic correctly then they may create bugs when they modify the code
4. it causes the code to match the most simple written algorithm less well.

It is easier code to read if it is general code that is relatively simple, but which can pass many different test cases.

Sometimes I have early return statements if I know that subsequent code would fail or not be executed. It keeps the later code free of a bunch of ugly statements intended to prevent errors (e.g., trying to access a non-existent list element). In my view such return statements are perfectly unambiguous. If you get to that spot in the code, nothing else will happen.

I believe the only major case of ‘implicit logic’ as you call it is when we create the FFP and SP lists. This function is thoroughly tested and has not changed for a long time. I’m also not sure of a better way to write it as we have to scan the reaction list until we find the proper type for the benchmark process. This function also assembles the fast and slow lists, so we have to check every reaction. I’m also not sure I would call it implicit. The only ‘implicit’ logic I see is when we find the first completely negligible process pair, we quit scanning processes because the remaining processes are guaranteed to be negligible because the list of process pairs is sorted and we are scanning rates in descending order.

Think about line 371 for FQPs

Not sure what this means. I think it’s related to whether we want to keep FQPs and FFPs together, but I’m not sure. Unless you can give me more concrete direction, I’m not doing anything with it.

for j in xrange(len(ascend\_FFP\_list)): 🡨 please change the index to rxn rather than or rxn\_j.

There are several examples of this. How aggressive should I be in making these changes?

\* I find the "find\_process\_pairs" and tg.EF\_indices\_dict  confusing. I will probably ask for this to be changed.

Need more info.

**Outstanding**

I don't like the wording of  local\_snapshot\_idx because it is not local to that function. How about just  “snapshot\_idx” or “current\_snapshot”

If your idea is that tg.current\_snapshot += 1  would be an *absolute* counter, then it should be part of the snapshots module and maybe be called sg.total\_snapshots since that would keep track of non-throttled snapshots also.

Your print statement would then call that variable in addition to sg.kmc\_time.

I think this would be a better way of keeping track of snapshots, I think you will agree.

Change oEF\_TOF\_list to just oEF\_list, and uEF\_TOF\_list to just uEF\_list. We should remove the “TOF” because that’s a “bad habit” to call it TOF when it’s really an EF. KMCOS should not call it TOF either, but unfortunately Max did use that variable name. Created bad habits in Tom’s nomenclature.