96 to 384 Cherry Picking

Stefan Golas

July 28 2022

Motivation

We want to compile a worklist into a set of time-efficient aspiration and dispense commands. To do this we have to identify sets of aspiration and dispense pairs where the wells for each aspiration and dispense (**a.d.**) pair lie in the same column. This way we can execute a list of multiple **a.d.** pairs simultaneously using the 8-channel pipetting arm.

An **a.d. list** is a 2-d list containing an aspiration list and a dispense list. Each **a.d. list** is composed of wells e.g. **(TargetPlate1, 4)** where each aspiration well is matched to its dispense well counterpart by position. An **a.d. list** can not exceed length n=8 for our purposes.

a.d. list := [[(TargetPlate, x), …]^n, [(SourcePlate, y), …]^n]

a.d. pair := (TargetPlate, x), (SourcePlate, y)

The list is extended when we add an additional pair. Every additional pair has to belong to the **worklist map**, an adjacency list of well-to-well transfers. We also remove the pair's source well from the worklist map for that target well when we add a pair.

https://en.wikipedia.org/wiki/Adjacency_list

This is the data type we are using to represent the worklist map.

wl = worklist_96('061322_32Seqs.csv')

wl is the **worklist map**, a dictionary mapping each target well to a set of source wells, e.g. $x_1 \rightarrow [y_1, y_2, ..]$ or in Python notation {target_well$_1$: [source_well$_1$, source_well$_2$, ..], ...}

$x_1 \rightarrow [y_1, y_2, ..]$

Add pair (x_1, y_1) to **a.d. list**

$x_1 \rightarrow [y_2, ..]$

When $x_1 \rightarrow [\ ]$ we delete x_1 as a key from the worklist map.

target_cols = get_96wp_columns([target_plate], 0, 1)
source_cols = get_columns_from_deck(source_plates, carrier_max_plates = 4)

source_cols and **target_cols** are 2-dimensional lists where each element is a list representing a column of wells, for every column from source or target plates. We treat all in-line columns from separate plates on the same carrier (e.g., the first column of every plate on the same carrier) as one column. This reflects the ability of the pipetting arm to pipette from these positions simultaneously.

For target columns of length n and quantity m:
target_cols := [[(TargetPlate, 0)^0, ... (TargetPlate, n)]^n, …]^m

For every combination of **source_col** and **target_col**, we expose only the links in the worklist map where each well belongs to its respective set in the intersection of **wl & (source_col | target_col)**, i.e., either source_col or target_col.

dispense_lists_from_wl(current_target_dict, target_cols, source_cols)
Now we have everything we need to compile a time-efficient set of a/d commands. Let's cover some basic concepts to understand how we do this.

The first concept is that for every pair of source and target columns, plus a worklist map from target to source wells, we can identify all of the column-wise matches between sets of a/d commands.

Therefore, our compiler will be a nested loop over every combination of target and source columns. Inside the nested loop, a function will take the combination of columns plus the worklist map as input, and output all of the column-wise matched a/d pairs for that combination.
Pseudocode #1:

```
total_list = []
worklist_map = read_worklist()
for tcol in target_cols:
    for scol in source_cols:
        ad_lists = enumerate_ad_lists(tcol, scol, worklist_map)
        total_list = total_list + ad_lists
```

enum_disp_from_cols(col_target_dict, tcol)

Here is the function for enumerating all a/d lists from the combination of source and target columns. Instead of directly providing the source column as input, we have simply created a new worklist map (**col_target_dict**) that provides only the source wells from the given column by doing a set intersection.

This function will then call a function that returns exactly one column-wise a/d pair, and returns this along with an updated worklist in which the members of that a/d pair are removed. This way we can check whether there are any remaining members of the worklist to know whether to move on or not.

disp_poss_from_dict(col_target_dict, tcol)
This function returns exactly one columnwise dispense pair along with an updated worklist map with the corresponding dispense pair removed. We loop over this function until the worklist is empty. Then, we move onto the next pair of source and target columns, and do the same. At 23 lines, this is the largest function in the script, although the details are mostly obvious.

Now we have accumulated a time-efficient set of aspiration and dispense pairs. But we can still apply further optimizations. Some of our pairs use 4 or less channels at a time. We can combine these pairs into a single pair, as long as the pair elements are shorter than length 8. Although we can't perform the commands in these combined pairs at the exact same time, we can pick up tips and eject tips for both operations at the same time. This will save a lot of time.

list_combine_disp(dl)
There are a lot of different heuristics we can use to combine aspiration and dispense pairs for a more time efficient set of commands. The heuristic in this script is very simple and not optimal, but still improves significantly over the original (Venus/ Hamilton) run-time.