blocking-explanation

Zhiwei Fan zfan29@wisc.edu
Lingfeng Huang lhuang58@wisc.edu
Fang Wang fwang64@wisc.edu

November 13, 2016

Questions

Question 1

How did you develop the final blocker? What blocker did you start with? What problems did you see? Then how did you revise it to come up with the next blocker? In short, explain the *development process*, from the first blocker all the way to the final blocker (that you submit in the Jupyter file).

We first decided to start the blocking procedure from some *simple blockers* with some *simple attributes* by appying some *simple strategies*. We first used the *attribute equivalence blocker* to filtered out those tuple pairs that didn't have the same number of pages on attribute *pages*. Later on, we decided to perform further *attribute equivalence blocking* on attributes *publishedYear*, *publishedMonth*, *publishedDay* together (*qualified tuple pairs* would *agree on all of these three attributes*). But we found that applying *strict equivalence blocking* on these three dates attribute would introduce the concern that many tuple pairs that should actually match each other were filtered out (*over-filtering*). This observation could be explained by the fact that some *dates attributes might be missing or dirty* and thus mis-filtered those tuple pairs that should actually match each other. In addition, we later on learned that *it's not necessary for the same books to have the same publication date. Even for the books of the same edition, the publication date could vary due to the fact that minor revision could be made to the books of the same edition and these books with different revisions could be re-published on different dates.* In order to mitigate the *mis-filtering* issue, we decided to use additional *edit distance blocker* on attributes *title*, *authors*, *publisher* with relatively *loose requirements* (similarity of two tuple pairs on *only* one of these attributes greater or equal than 0.5 would be *qualified* and remained for the *entity matching* procedure later). In this manner, we expected those tuple pairs would not match each other would be filtered out while *mis-filtering* on actually *matched* tuples would not happen frequently. However, after running the *edit distance blocker*, we soon observed that the procedure was way to slow (at the time when we checked estimation time in terminal, it was more than 10 hours and kept increasing) and realized that this was due to the fact that edit distance computation was *very expensive* (applying edit distance on tuple pairs would not be realistic since edit-distance computation on each tuple brings complexity $O(n^2)$) and applying edit distance on the original tuple pairs would be unrealistic since it was unacceptably time-consuming. We then switched to *loose Jaccard blocker on attributes title, authors, publisher* (tuple pairs that do not agree on any of these three attributes will be filtered out). But even Jaccard blocker was slow: we haven't seen *any* obvious progress after waiting for long time. So we *narrowed* the blocking condition again: doing Jaccard blocker *only* on *publisher*. Bug again, disappointed by the fact that *Jaccard blocker on even only publisher* didn't show any progress after 20 minutes, so we again changed our strategy: using the first and last three digits of isbn to perform blocking. After many attempts, we finally concluded that putting the complete blocking logic in one blocker is not an ideal way and decided to use three blockers: *equivalence blocker on attribute pages*, *strict equivalence blocker on attributes publishedYear, publishedMonth and publishedDay*, *partial isbn blocker* to filter those *unqualified tuple pairs* and avoid *over-filtering* at the same time.

Question 2

If you use Magellan, then did you use the debugger? If so, where in the process? And what did you find? Was it useful, in what way? If you do not use Magellan, you can skip this question.

We used the debugger every time after we have performed some blocking on tuple pairs (*every time after performing a blocker on table A and table B*. We used the debugger to check how many potential tuple pairs would be

discarded by our blocker and adjust our blocking strategies (for example, adding additional blocker after trying *strict equivalence blocker on attributes publishedYear, publishedMonth and publishedDay* as described in **Question 1**). By observing the output of the debugger, we could have a general sense whether we have *thrown away* two many tuple pairs that should actually match. So we think the debugger is very useful in making strategies of blocking and giving suggestions on adjustment of blocking strategies. In addition, after *loosing* the blocking requirement, we have still observed many *uncaught* tuple pairs that should match according to the output of debugger. The observation told us the data was *dirtier* than we thought before (*missing values and wrong attribute values appear frequently*. In real life, this observation would lead to motivation of *data re-cleaning*. Due to the time limitation, we haven't applied this good practice though. One additional interesting observation was that even when we tried to match tuple pairs using isbn (which we didn't actually use in our blocking process but tried for curiosity of the matching results using isbn), debugger still showed that there were many tuple pairs were filtered out that should actually match each other.

Question 3

How much time did it take for you to do the whole blocking process? We have spent most time trying different blocking strategies. Among all of the strategies we have tried, we have found that applying *edit distance* and *Jaccard* (very likely including other sequence and set methods) on tuple pairs of large number would be extremely slow (we have conducted experiments on these two methods and the procedure took us around 10 hours). We have also spent time analyzing the output of debugger and kept updating our blocking strategies. The final blockers we have come up with would finish the whole blocking procedure in less than 10 minutes (using four cores). The whole procedure took us about 4 days.

Question 4

Report the size of table A, the size of table B, the total number of tuple pairs in the Catersian product of A and B, and the total number of tuple pairs in the table C.

Size of *table A*:
5279 tuples

Size of *table B*:
3785 tuples

Total number of tuple pairs in the Cartesian product of A and B:
$5279 \times 3785 = 19981015$ tuple pairs

Total number of tuple pairs in *table C*:
71690

Question 5

Did you have to do any cleaning or additional information extraction on tables A and B?

When we tried to read *table A* and *table B*, we have encountered the *error: the selected attribute (id) is not qualified as key*. We then went back to check the source csv file corresponding to the two tables, finding that some tuples are missing and thus some *ids* are also *missing*. The reason behind these missing tuples could be that the *spider* we have used in stage 1 automatically filtered these tuples since the structure of their attributes were not compatible to those rules (inferred from the source websites) we defined in our spider. Also, we have found out that the attribute *publishedYear* was recognized as *object type* in *table A* but recognized as *numerical type* in *table B*. We manually *reset* the id attribute in both tables and covert the type of *publishedYear* in *table A* to *numerical type* from *object type* in order to perform comparison with *publishedYear* in *table B*. We have also considered doing cleaning on these attributes found to have missing values in stage 2 (e.g., pages, publishedYear, publishedDay, publishedMonth), but then we decided not to do the extra cleaning on these attributes after careful consideration of the tradeoff between the complexity, extra workload and time needed for cleaning and the benefit of cleaning would bring to us.

Question 6

Did you run into any issues using Magellan (such as scalability?). Provide feedback on Magellan. Is there anything you want to see in Magellan (and is not there)? If you do not use Magellan, you

can skip this question.

Fortunately, we didn't really run into any serious issues using Magellan (functionalities we have encountered so far are friendly to use and worked very well along with our *blocking procedure*). One of the concerns we have is that when we were doing *multi-attribute blocking* (defined in our black-box function comparing several attribute at the same time), we found that the blocking speed was too slow considering the fact the number of tuples was quite small in our task. Thus we think further optimization of the computation is possible such as *parallel computation* as addressed in *Additional feedback session*. The other inconvenience we have found is that we didn't find a convenient way to perform further blocking on the output of one block operation (e.g., after perform attribute equivalence blocking on source tables, we want to consider *only* these tuple pairs in the candidate set returned by the previous blocking operation). We have seen the function named *block-tuples* in documentation and attempted to use it, however, due to lack of clear examples and documentation, we didn't make to use it finally (maybe the function was not even implemented for this purpose originally). We think at least one simple example for each command should be given so that users could easily use more functionalities without too much confusion. And providing a function to allow users to perform further block directly on the output of previous blocking would be very helpful.

Additional feedback

When we were doing *attributes equivalence blocking*, we have noticed that ***NOT*** *all the cpu resource was fully utilized* (we have 4 cores available while only one core is being used). We think that the support for *automatic parallel computation* should be added due to the natural property of many blocking procedure. For example, after using hash function to partition the table A and table B into 4 partitions (A1, B1), (A2, B2), (A3, B3), (A4, B4), the *blocking* between Ai and Bi could be processed asynchronously. And currently there are quite a few python modules supporting automatic parallel computation in system with multiple cores and clusters (such as *Parallel Python*) and parallel computation support could be relatively easily added by using those modules. We have noticed that *parallel computation* is mentioned to be supported by Magellan in the documentation, however, it took us quite a while to figure out how to *trigger parallel compuation*. We think it not very reasonable to require entering all parameters for a single function while the only setting we want to change is the number of cores to be used. So we think the functionality allowing setting properties such as *number of cores to use* separately should be allowed (e.g., em.setNumberOfCores(coreNum)).