PharmaSUG 2018 - Paper AD34

Metabolite Identification and Characterization by Mining Mass Spectrometry Data with SAS® and Python

Kristen Cardinal, Colorado Springs, Colorado, United States
Hao Sun, Sun Prairie, Wisconsin, United States

ABSTRACT

Data analysis, visualization and reporting of preclinical and clinical drug metabolism assays were automated with a SAS®-based application, AIR Binder (PharmaSUG 2017, MWSUG 2017). Data mining using SAS® in combination with Python was also explored for a specific dataset type in drug metabolism, the mass spectrometry data for metabolite identification and characterization. We developed an application, LEO, to generate solutions for relatively challenging metabolite characterization of peptides and antibody-drug conjugates, which was built on an algorithm to search accurate mass change using three main components: *linearization* for cleavage reactions, *extension* for conjugation reactions, and *oxidation* for oxidation-reduction reactions (SASGF 2018). We further extended applications of LEO to a wider chemical space, for metabolite identification of both small and non-small molecules. The scoring functions in LEO were optimized by integrating analysis of fragmentation patterns, applying searching and iteration algorithms on fragments, and generating comprehensive solutions for recognizing aligned metabolite and fragment patterns. In addition, fragmentation patterns of parent compound were integrated as an option to refine solutions. Overall, LEO facilitated the process of identifying and characterizing "difficult" metabolites in drug discovery and development.

INTRODUCTION

Data analysis in the area of drug metabolism is diverse, from simple enzyme kinetics fit with non-linear equations to complicated multi-compartment pharmacokinetics modeled by differential equations. We reported a SAS®-based application, AIR Binder, to automate data analysis and visualization on various assays in drug metabolism (PharmaSUG 2017, MWSUG 2017). The similarity of diverse datasets reported in AIR Binder was their quantitative nature, regardless of instruments and software programs that generated them. Mass spectrometry has been routinely used nowadays to generate data for quantitation purpose including datasets for in vitro ADME assays, pharmacokinetic analysis, and metabolite profiling. As long as they are quantitative data points, they can be easily imported into AIR Binder for automation using SAS® macros, for analysis with various linear, nonlinear and mixed models, and for visualization with powerful ODS graphics. However, the datasets from mass spectrometry for metabolite identification and characterization are different, which are mainly qualitative data such as mass and fragments used for pattern analysis and recognition.

Molecular ions of metabolites from mass spectrometry are relatively simple to identify for small molecules, but may be complicated for peptides and antibody-drug conjugates. We reported a method in LEO to provide solutions for ranking most probable molecular ions by mechanisms (SASGF 2018). Accurate mass change after metabolism was predicted and searched by considering biotransformation mechanisms such as cleavage reactions ("***L***inearization"), conjugation reactions ("***E***xtension"), and oxidation-reduction reactions ("***O***xidation"). Once molecular ion solutions were generated, scoring functions were used to identify most probable combinations by comparing fragment patterns. For small molecules, fragment pattern analysis is straightforward in most cases that can be accomplished by chemical structure drawing programs manually. However, for non-small molecules and relatively complex small molecules, it is more time-consuming to manually match fragments. Especially, it is time-consuming to identify "difficult" metabolites. For current work, the algorithm used to collect solutions for molecular ions of metabolites is used similarly to find solutions for fragments. The patterns of both metabolites and fragments are analyzed and scored to rank solutions. The raw datasets for LEO include molecule ions and fragments of both parent compound and metabolites (Figure 1). A "smart" pattern analysis of these datasets are essential for fast and accurate metabolite identification and characterization.

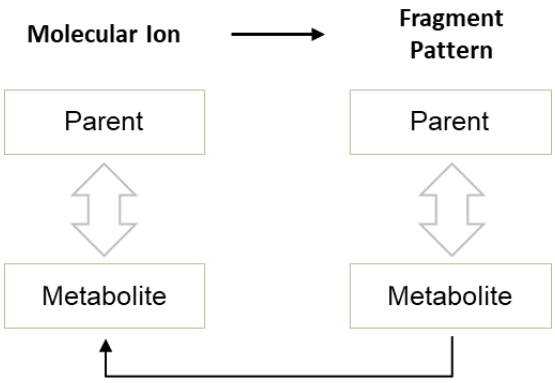

Figure 1 Datasets for metabolite identification and characterization

The process to generate three searching components of LEO including the compound-dependent “L” and database “E” and “O” (lists and dictionaries) in Python were described previously (SASGF 2018). For current process, two essential functions in LEO are stored in searching/iteration and scoring modules (Figure 2). Molecular ions of metabolites and associated top fragments by abundance are the input parameters for the searching/iteration module. The solution list for a metabolite is generated according to algorithms, together with solution lists for fragments. It is more flexible in the current version to define L, E, and O components based on the molecular structure of a molecule. All solution lists for fragments are pooled to generate an overall solution table, and then compared with the metabolite solution list using the “scoring module 1”, to rank solutions for metabolite identification and characterization. As an option, the “scoring module 2” is used to compare the fragmentation patterns of a parent compound and metabolites. Although fragment patterns of the parent compound and metabolites may not be consistent, which depend on mechanisms of biotransformation, still it provides useful information for characterizing majority of metabolites of a small molecule drug.

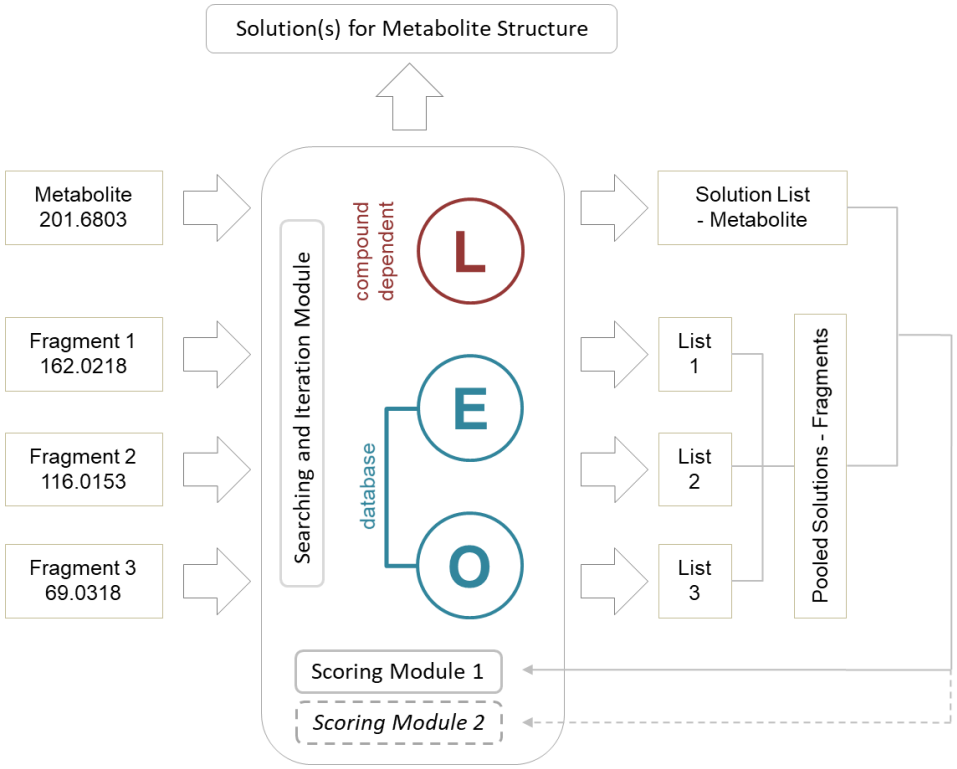

Figure 2 Searching and scoring in LEO for metabolite identification and characterization

SEARCHING AND ITERATION IN LEO

In vivo drug metabolism and biotransformation may involve multiple steps of reactions, which makes the searching and iteration process challenging for identification and characterization of a metabolite. Theoretically, unlimited steps of reactions can be searched with accurate mass change calculated in LEO, but the number of loops would be astronomical, as well as computing and data analysis time. The original purpose of developing LEO was to help experienced metabolite identification scientists find solutions faster and analyze mass spectrometry data more efficiently. Users can make decisions based on their understanding and expectation of specific metabolic pathways, a targeted searching approach. For example, if more conjugation reactions are involved, the scheme "*a*" as follows is a good starting point. On the other hand, if unknown mechanisms are expected, an offset of each atom type in the scheme "*b*" is a better choice. Some compound contains special atoms such as a fluorine atom, and thus the scheme "*c*" should capture fluorine-related reactions.

```
a.  leo_search = [linearization, extension1, extension2, extension3]
b.  leo_search = [linearization, extension1, extension2, carbon, hydrogen, oxygen]
c.  leo_search = [linearization, extension1, carbon, hydrogen, oxygen, fluorine]
```

The function massLEO in Python as follows is designed to execute iterations for flexible input of parameters in LEO. Values derived from the list *leo_dictx* such as mass and mass strings are added to those of *leo_dict*. The ppm (parts per million) values are also calculated during each iteration. For example, strings from a previous list are read in directly with new strings added by looking up mass dictionaries in LEO. These string combinations are separated by vertical bars. Numbers and strings from a new iteration are added to the lists *leo_str*, *leo_ppm*, and *leo_mass* with the APPEND function. The string "LEO" is used to represent an empty component.

```
def massLEO(leo_dict, leo_str, leo_ppm, leo_mass, leo_dictx):
   for i in xrange(len(leo_dict)):
       for j in xrange(len(leo_dictx)):
           mass = leo_dict[i] + leo_dictx[j]
           if mass != 0:
                ppm = 1000000*(leo_mass-mass)/mass
           else:
                ppm = 0
           mass_str = str(leo_str[i]) + " | " + str(massdict[leo_dictx[j]])
           leo_dict.append(mass)
           leo_ppm.append(ppm)
           leo_str.append(mass_str)

mass_list = [0]
mass_str = ['']
mass_ppm = [0]
for i in xrange(len(leo_search)):
        massLEO(mass_list, mass_str, mass_ppm, ionmass, leo_search[i])

outfile = open("leo_" + str(ion) + "_" + iontype + ".txt", "w")
for i in xrange (len(mass_list)):
        if mass_list[i] > lower_ppm and mass_list[i]  < upper_ppm:
                outfile.write(str(mass_list[i]) + "," + str(mass_ppm[i]) + ","
                        + str(mass_str[i]) + "\n")
```

With the searching and iteration algorithm in LEO, duplicated solutions may be generated during each round of iteration. For example, if two "*E"* lists are selected, different order of "E1" and "E2" generate duplicated solutions ("*a*" and "*b*" as follows) due to shared conjugation reaction database in "E". In addition, the empty components that are displayed with "LEO" may generate duplicated solutions ("*c*"-"e"). They can be removed by the SAS component of LEO for further data analysis, scoring and ranking.

```
a. 201.6803,-2.83,L1 | Cysteine | Acetylation
b. 201.6803,-2.83,L1 | Acetylation | Cysteine
c. 201.6803,-2.83,L1 | Acetylation | Cysteine | LEO
d. 201.6803,-2.83,L1 | Acetylation | Cysteine | LEO | LEO
e. 201.6803,-2.83,L1 | Acetylation | Cysteine | LEO | LEO | LEO
```

DISPLAYING AND RANKING SOLUTIONS IN LEO

The raw output data from Python are analyzed with the macro LEO in SAS® as follows. The solution combinations as separated by vertical bars are read in as a single long string, which is sorted first to remove duplicates using the NODUPKEY option with the SORT procedure. Each individual component of a solution is then stored in an array called *leoarray*. Duplicates are further removed by the IF statement that selects the rows with all searching component columns "filled" (including those with "LEO" empty strings). The list of components are sorted with a "bubble sort" algorithm (SASGF 2018), and duplicate solutions are removed. The initial score of each solution is calculated in favor of simplicity of combinations (less components). Top solutions are then selected for metabolites and fragments, followed by scoring and ranking.

```
%macro leo (infile=in, out=out, leo_group=in, top=in);
   data &out.leo;
      infile &infile dsd dlm=',';
      input mass ppm leostr :$830.;
      format ppm 4.2;
   run;

   proc sort data=&out.leo out=&out.leo_nodup nodupkey; by leostr; run;

   data &out.leo_all (drop=i);
      set &out.leo_nodup;
      length leoarray1-leoarray&leo_group $83;
      array leoarray(&leo_group) $;
      do i=1 to dim(leoarray);
           leoarray[i]=scan(leostr,i,'|');
      end;
      if leoarray&leo_group ne '';
   run;

   data &out.leo_allsorted (drop=i j leotemp);
      set &out.leo_all;
      array leo(&leo_group) leoarray1-leoarray&leo_group;
      do i=1 to dim(leo);
           do j=1 to &leo_group.-i;
             if leo(j)>leo(j+1) then do;
                leotemp=leo(j);
                leo(j)=leo(j+1);
                leo(j+1)=leotemp;
             end;
           end;
      end;
   run;

   proc sort data=&out.leo_allsorted out=&out.leo_unique nodupkey;
             by leoarray1-leoarray&leo_group;
   run;

   data &out.leo_rank (drop=leoscore1-leoscore&leo_group mass leostr i);
      set &out.leo_unique;
      array leo_prescore(&leo_group) leoarray1-leoarray&leo_group;
      array leoscore(&leo_group) leoscore1-leoscore&leo_group;
      score=0;
      do i=1 to dim(leo_prescore);
             if strip(leo_prescore(i)) eq "LEO" then leoscore(i)=0;
             else leoscore(i)=1;
             score=score+leoscore(i);
      end;
      leo_solution=tranwrd(leostr,'LEO','');
      leo_solution=tranwrd(leo_solution,'| ','');
   run;

   proc sort data=&out.leo_rank; by score; run;
   data &out (keep=ppm leo_solution); set &out.leo_rank(obs=&top); run;
   proc print data=&out; run;
%mend;
```

CONCLUSION

LEO is an application we developed to facilitate metabolite identification and characterization process in drug discovery and development. It demonstrated significantly enhanced productivity to find solutions for "difficult" metabolites and "difficult" molecules. SAS® in combination with Python provided convenience to write searching and iteration algorithms for pattern analysis of mass spectrometry data.

ACKNOWLEDGMENTS

The authors would like to thank Rich Voorman, Xiumin Liu, and Bob Greene for their discussion on and involvement of testing AIR Binder and LEO. The authors are also grateful to feedback from our clients and friends across pharmaceutical and biotech companies, especially, Steve Alley and Anthony Lee from Seattle Genetics.

REFERENCES

Sun, H., Cardinal, K., Kirby, C., and Voorman, R. 2017. “AIR Binder: an automatic reporting and data analysis SAS® application for cytochrome P450 inhibition assay to investigate DDI.” *PharmaSUG* Paper PO09: 1-8.

Sun, H., Cardinal, K., and Voorman, R. 2017. “AIR Binder 2.0: A dynamic visualization, data analysis and reporting SAS® application for preclinical and clinical ADME assays, pharmacokinetics, metabolite profiling and identification.” MWSUG Paper PH04: 1-26.

Sun, H. and Cardinal, K. 2018. “Mining metabolites of peptides and antibody-drug conjugates from mass spectrometry data using SAS® and Python.” SASGF Paper 2510: 1-6.

CONTACT INFORMATION

Hao Sun, PhD
118 Cobham Ln
Sun Prairie, Wisconsin 53590
United States

Email: CYP2F1@gmail.com
Phone: 801.541.0884

Current Address: Hao Sun, PhD | Seattle Genetics | 21823 30th Dr SE, Bothell, WA 98021 | Office: 425-527-4681 | Email: hsun@seagen.com