A Comparison of Reaction Probability Combinations

Eric Chanowich
NOM Research Group
University of Notre Dame

Abstract. In this document, two versions of reaction probability combinations are presented and their validities are discussed. The first reaction probability combination method is presented in Pascal by Dr. S. Cabaniss of the University of New Mexico. The second method is presented in Java by Eric Chanowich, an undergraduate student at University of Notre Dame.

I. Background

10 reactions probabilities are calculated for each molecule. The reaction probabilities, however, represent the probability of only one reaction occurring or not occurring. Therefore, we seek some way to merge the reaction probabilities, thus finding the combined chances of a given reaction occurring while "competing" with other reactions. The ultimate goal is to arrange the combined probabilities such that a random number between 0 and 1 may be generated and checked against these probabilities to determine if a reaction should occur and, if so, which reaction should occur.

II. The Reaction Probability Combinations of Dr. S. Cabaniss

The process of combining probabilities to choose whether on not a reaction occurs is composed of three procedures in Dr. Cabaniss's Pascal code. The first procedure, *ProbCalc* (Fig. 1), takes a molecule, an empty array of probabilities, and ΔT, the time step size. *ProbCalc* sends the molecule and ΔT to each of the individual stand-alone probability calculating procedures. The stand-alone reaction probability of each reaction type is stored in a corresponding location (1-10) of the array of probabilities. Finally, all of the stand-alone probabilities are summed and this value is placed in location 0 of the array of probabilities.

Fig 1.

```
Procedure ProbCalc(Var MoleStr : Molecule;       {molecule to be considered}
                   Var ProbList : RxnProbs;      {probabilities to calculate}
                   DeltaT : real);               {time interval }

Var i : integer;                        {counts reactions}
    CumeProb : real;                    {total  probability of a reaction}

begin
  Problist[1] := EsterCondProb(Molestr, DeltaT);
  Problist[2] := EsterHydrProb(Molestr, DeltaT);
  Problist[3] := AmideHydrProb(Molestr, DeltaT);
  Problist[4] := MicrobUptkProb(Molestr, DeltaT);
  Problist[5] := DehydrateProb(Molestr, DeltaT);
  Problist[6] := StrongOxProb(Molestr, DeltaT);
  Problist[7] := MildOxProb(Molestr, DeltaT);
  Problist[8] := AlcOxProb(Molestr, DeltaT);
  Problist[9] := AldeOxProb(Molestr, DeltaT);
  Problist[10] := DecarboxProb(Molestr, DeltaT);

  CumeProb := 0;
  for i := 1 to NumRxns do
      CumeProb := CumeProb + Problist[i];
  ProbList[0] := CumeProb;

end;                                          { end of procedure ProbCalc }
```

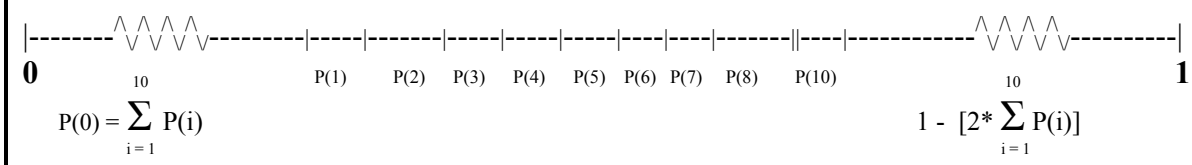
The next function, *CumeProb* (Fig. 2), essentially generates a random number (*ThisProb*) that is distributed normally from 0 to 1, as the sum of all the probabilities must be less than 1 and is assumed to be less than 0.1. The random number is the sequentially check against the sum of each of the array locations from 0 to 10. Recall, from the previous procedure, *ProbCalc*, that array position 0 is already the sum of all of the probabilities. Essentially, this procedure evaluates all of the probabilities as if they are on a number line as show in Fig. 3.

Fig. 2

```
Function CumeProb(N : integer; Var EventList : RxnProbs) : integer;
Var     ThisProb : Double;          {a uniform deviate}
        SumProb  : Double;          {Summed probability of events}
        i        : Integer;         {counter index}

begin
        i := 0;                     {event index to first event}
        SumProb := 0;               {therefore no probability}
        ThisProb := Unidev;
        Repeat
              SumProb := SumProb + Eventlist[i];
              i := i + 1;
           until ((ThisProb < SumProb) or (i = N));
        Result := i;
        If (ThisProb > SumProb) then Result := 0;
end;                          { end of CumeProb}
```

Fig. 3



* Note: In this diagram, P(9) is omitted to illustrate a reaction with zero probability.

As illustrated in Fig. 3, the virtual number line that is produced has 12 segments. The first segment is from 0 to the sum of P(1) through P(10). Then, each of the 10 stand-alone probabilities are added in order as segments on the number line with the exact same size as their stand-alone probability. Finally, the last segment is what ever remain from the 11th segment to 1, which ends of being of size 1 minus 2 times the sum of all of the stand-alone probabilities.

The final procedure, *ReactMol* (Fig. 4), essentially combines the two procedures that were previously discussed. *Problist* is sent to *CumeProb* and *ReactNumbr* is returned. *ReactNumbr* is then used to select one of the reaction procedures. If *ReactNumbr* is equal to 0, then no reaction occurs and the *ReactMol* procedure simply exits. If *ReactNumbr* is greater than 0 (and less than or equal to 10, of course), then the procedure of the given reaction is called. When the reaction procedure has returned, the probabilities of the new molecule are calculated. Finally, the reaction number is recorded in a counter and the *ReactMol* procedure exits.

Fig. 4

```
Function ReactMol(Var MolStruct : molecule;     {structure of molecule to react}
                  Var ProbList : RxnProbs;      {list of reaction probabilities}
               const DeltaT : real) : integer;  {length of time interval in hours}
Var
   ReactNumbr : integer;        {number of reaction which occurs}
   ReactOK    : Boolean;        {Did the reaction occur?}
   Empty      : integer;        {location of empty molecule for splitting}
   BiMol      : integer;        {location of molecule to participate in }
                                { a bimolecular reaction (ester condensation)}
begin
  Result := 0;          {set function to zero, i.e., no reaction occurs}

  { if no structure exists for this molecule, then end routine, return 0}
  If (MolStruct.Exist = False) then Exit;

  {now determine if a reaction can occur by stochastic procedure }
  ReactNumbr := CumeProb(NumRxns, Problist);   {first, select a reaction}

  case ReactNumbr of
  0 : Exit;                       {if no reaction occurs, then return 0}
  1 : begin                       {Ester Condensation}
        {first, find a molecule to react with}
        BiMol := Random(NumMol) + 1;
        {then check to be certain it exists and can react}
        if (MolecSet[BiMol].Exist and (MolecSet[BiMol].Groups[Alcohols] >0))
           then EsterCond(MolStruct, MolecSet[BiMol])
              else ReactNumbr := 0;
      end;
  2 : begin                       {Ester Hydrolysis}
        Empty := FindEmptyMol(MolecSet, NumMol);
        if (Empty > 0) then EsterHydr(MolStruct, MolecSet[Empty]);
      end;
  3 : begin                       {Amide Hydrolysis}
        Empty := FindEmptyMol(MolecSet, NumMol);
        if (Empty > 0) then AmideHydr(MolStruct, MolecSet[Empty]);
      end;
  4 : ReactOK := AldeOx(MolStruct);      {Microbial Uptake}
  5 : ReactOK := Dehydrate(MolStruct);   {Dehydration}
  6 : begin                              {Strong Oxidations}
        Empty := FindEmptyMol(MolecSet, NumMol);
        if (Empty > 0) then StrongOx(MolStruct, MolecSet[Empty]);
      end;
  7 : ReactOK := Mildox(MolStruct);      {Mild Oxidation}
  8 : ReactOK := AlcOx(MolStruct);       {Alcohol Oxidation}
  9 : ReactOK := AldeOx(MolStruct);      {Aldehyde oxidation}
  10: ReactOK := DeCarbox(MolStruct);    {Decarboxylation}
 end;                   {end of cases }

  {Now compute the reaction probabilities for the newly-modified molecule}
  ProbCalc(MolStruct, ProbList, DeltaT);

  {and finally return number of the reaction that occurred}
  Result := ReactNumbr;

end;                    {end of ReactMol}
```

III. The Reaction Probability Combinations of Eric Chanowich

Again, we seek to normalize a group of stand-alone probabilities into a group of numbers on a number line from 0 to 1. We assume that all of the stand-alone reaction probabilities have been correctly computed and are being passed to the *probNormalize* (Fig. 5) method as *originalArray*. *originalArray* is an array of size *ReactionSize*, a global variable that is the

number of reactions. The 10 stand-alone probabilities are each stored in a corresponding array location in *orginalArray* from 0 to 9.

The calculation of the combined reaction probabilities is essentially accomplished in two *for* loops. The first loop iterates through each of the stand-alone reaction probabilities in *originalArray* and calculates the sum of each of the stand-alone probabilities, as well as the product of their complements (1-P). The formulas for calculating *sumOriginal* and *complementProduct* can be found in Fig. 6a.

Fig. 5

```
ReactionSize = 10;

protected void probNormalize( double originalArray[]) {

  double sumOriginal = 0;     // Sum of the original probabilities
  double sumNormalized = 0;   // Sum of the normalized probabilities
  double complement = 1;      // Product of (1-complements)

  double normalizedProb[] = new double[ReactionSize+1];

  for ( int count = 0; count < originalArray.length; count++ ) {
    sumOriginal += originalArray[count];
    complementProduct *= (1 - originalArray[count]);
  }

  normalizedProb[0] = 0;

  for ( int count = 0; count < sizeOfArray; count++ ) {
    sumNormalized += ((originalArray[count] * (1 - complementProduct)) / sumOriginal);
    normalizedProb[count+1] = sumNormalized;
  }

} // end probNormalize
```

The second *for* loop uses the information attained in the first *for* loop to fill in the *normalizedProb* array according to the formulas in Fig. 6b.

Fig. 6a

$$sumOrginial = \sum_{i=0}^{9} originalArray[i]$$

$$complementProduct_0 = 1$$
$$complementProduct_i = complementProduct_{i-1} * (1 - originalArray[i-1])$$

Fig. 6b

$$normalizedProb[0] = 0$$

$$normalizedProb[x] = \sum_{i=1}^{x} originalArray[i-1] * (1 - complementProduct_{10}) * (1 / sumOriginal)$$

IV. Conclusions

Let us investigate both formulas while considering three individual events A, B, and C where P(A), P(B), and P(C) are their respective stand-alone probabilities. P(N) is the

probability that none of the events will occur if they are all individual events. We can solve for the probability of event A when combined with events B and C, P'(A), and the combined probability of no reaction, P'(N).

Fig. 7

$$P(N) = (1 - P(A))*(1 - P(B))*(1 - P(C))$$

Dr. S. Cabaniss's Formula:

$$P'(A) = P(A)$$

$$P'(N) = 1 - P(A) - P(B) - P(C)$$

Eric Chanowich's Formula:

$$P'(A) = \frac{P(A) * (1 - P(N))}{P(A)+P(B)+P(C)}$$

$$P'(N) = 1 - \frac{(P(A) + P(B) + P(C)) * (1 - P(N))}{P(A) + P(B) + P(C)} = P(N)$$

Intuitively, if two or more reactions with probability greater than 0 exist (at least two probabilities must be great than 0 or there is no reason to combine them), we know that the probability of each individual reaction occurring will decrease from its stand-alone probability as the reactions must now "compete" with each other to occur. Of course, if the stand-alone alone probability of a reaction is 0, its combined probability will remain 0.

In Dr. Cabaniss's method, such is not the case. In fact, all of the individual reaction probabilities remain exactly the same as their "stand-alone" probability. While, the combined probability of no reaction meets our criteria of being less than the probability of any single reaction not occurring, it fails to reflect that the reactions are being combined and simply subtracts the sum of the stand-alone probabilities from 1.

In Eric Chanowich's method, the constraint that $P'(A) < P(A)$ is met. By adjusting Eric's first formula in Fig. 7, we can see that $(1\text{-}P(N)) / (P(A) + P(B) + P(C))$ must be less than 1 for this constraint to hold as P'(A) is equal to P(A) multiplied by the previous term. Fig. 8 illustrates the expansion of the term. We can clearly see after expanding that a critical part of eq. 1, shown in eq. 2, must be less than 0 for the term as a whole to be less than 1 and $P'(A) < P(A)$.

Fig. 8

1) $$P'(A) = P(A) * \frac{1 - (1 - P(A))*(1 - P(B))*(1\text{-}P(C))}{P(A) + P(B) + P(C)}$$

$$= P(A) * \frac{P(A) + P(B) + P(C) - P(A)P(B) - P(B)P(C) - P(A)P(C) + P(A)P(B)P(C)}{P(A) + P(B) + P(C)}$$

2) $$P(A)P(B)P(C) - P(A)P(B) - P(B)P(C) - P(A)P(C) < 0$$

Upon careful examination of the second equation of Fig. 8, we can see than P(A)P(B)P(C) – P(A)P(B) – P(B)(C) – P(A)P(C) < 0 is always true where at least two of the probabilities is greater than 0. P(A)P(B)P(C) must be less than at least one of the products of two probabilities, thus making the term less than 0 and make P'(A) < P(A), satisfying our intuitive constraint.

While Eric Chanowich's equations are not necessarily correct, they satisfy necessary constraints and they *could* be correct. The equations are an efficient way of combining reaction probabilities in the NOM simulation.