DECEPTIVELY SIMPLE:

HOW SOME CHEMINFORMATICS PROBLEMS CAN BE MORE COMPLICATED THAN THEY APPEAR

ROGER SAYLE

NEXTMOVE SOFTWARE, CAMBRIDGE, UK

7th RDKit UGM, Cambridge, UK, Thursday 20th September 2018

OVERVIEW

1. Calculating molecular weight.
2. Counting lines in a text file.
3. Determining percentages.

Cheminformatics in a nutshell

- “The greatest contributions of cheminformatics to drug discovery have been molecular weight and LogP.” – J. Andrew Grant.

- “All right, but apart from the sanitation, the medicine, education, wine, public order, irrigation, roads, a fresh water system and public health, what have Romans ever done for us?” – John Cleese, The Life of Brian.
- “In Italy for 30 years under the Borgias they had warfare, terror, murder and bloodshed, but they produced Michelangelo, Leonardo da Vinci and the Renaissance. In Switzerland, they had brotherly love – they had 400 years of democracy and peace, and what did that produce? The cuckoo clock.” – Orson Welles, The third man.

MOLECULAR WEIGHT #1: AMW

“calculated as the sum of the atomic weights of each constituent element multiplied by the number of atoms of the element in the molecular formula” – Wikipedia.

```
double mw = 0.0;
PeriodicTable *table = PeriodicTable::gettable()
for (ROMol::ConstAtomIterator atomIt = mol.beginAtoms();
     atomIt != mol.endAtoms(); ++atomIt) {
  int atNum = (*atomIt)->getAtomicNum();
  mw += table->getAtomicWeight(atNum);
}
return mw;
```

$C_6Cl_6 = 6*12.011 + 6*35.45 = 284.77$

Molecular Weight #2: Implicit H

A distinction between computational chemistry and cheminformatics is the “implicit hydrogen”.

```
  double mw = 0.0;
  unsigned int hcount = 0;
  PeriodicTable *table = PeriodicTable::gettable();
  for (ROMol::ConstAtomIterator atomIt = mol.beginAtoms();
       atomIt != mol.endAtoms(); ++atomIt) {
    unsigned int atNum = (*atomIt)->getAtomicNum();
    mw += table->getAtomicWeight(atNum);
    hcount += (*atomIt)->getTotalNumHs();
  }
  if (hcount)
    mw += hcount * table->getAtomicWeight(1);
  return mw;
}
```

C_6H_6 = 6*12.011 + 6*1.008 = 78.114

Molecular Weight #3: Isotopes

When specific isotopes of individual atoms are specified/known, they should be honoured.

```
unsigned int atNum = (*atomIt)->getAtomicNum();
unsigned int iso = (*atomIt)->getIsotope();
if (iso != 0)
  mw += table->getMassForIsotope(atNum,iso);
else mw += table->getAtomicWeight(atNum);
```

$^{13}CH_4 = 13.00335483507 + 4*1.008 = 17.035$

Molecular Weight #4: Monoisotopic Mass

So far we’ve used conventional masses based upon weighted averages of natural abundance, that don’t necessarily correspond to a peak in a mass spectrum.

Monoisotopic masses use the most abundant isotope

```
calcAMW                                calcExactMW
mw+=table->getAtomicWeight(atNum);     mw+=table->getMostCommonIsotopeMass(atNum);
mw+=hcount*table->getAtomicWeight(1);  mw+=hcount*table->getMostCommonIsotopeMass(1);
```

AMW: $C_6Cl_6 = 6*12.011 + 6*35.45 = 284.77$

Monoisotopic $C_6Cl_6 = 6*12.0000000$

$+ 6*34.968852682$

$= 281.8131161$

Molecular Weight #5: Uncertainty

Atomic masses have two sources of uncertainty, experimental error and statistical abundance variation.

Dave Weininger’s amw.c tracks decimal places.

```
double mw = 0.0;
unsigned int prec = 9;
for (ROMol::ConstAtomIterator atomIt = mol.beginAtoms();
     atomIt != mol.endAtoms(); ++atomIt) {
  int atNum = (*atomIt)->getAtomicNum();
  mw += table->getAtomicWeight(atNum);
  prec = min(prec,table->getAtomicWeightPrecision[atNum]);  // Hypothetical
}
printf(“%.*f\n”,prec,mw);
```

$PbF_4 = 207.2 + 4*18.998403163 = 283.2$

MOLECULAR WEIGHT #6: BOUNDS

A better approach is to use interval arithmetic, determining upper and lower bounds on MW.

```
double mw_lower = 0.0;
double mw_upper = 0.0;
for (ROMol::ConstAtomIterator atomIt = mol.beginAtoms();
     atomIt != mol.endAtoms(); ++atomIt) {
  int atNum = (*atomIt)->getAtomicNum();
  mw_lower += table->getLowerAtomicWeight(atNum);  // Hypothetical
  mw_upper += table->getUpperAtomicWeight(atNum);  // Hypothetical
}
```

Monoisotopic mass of Crambin ($C_{202}H_{315}N_{55}O_{64}S_6$)

lower = 202*12.00000000+315*1.007825032230+55*14.003074004430
+64*15.994914619570+6*31.97207117440 = **4727.1409**1809498

upper = 202*12.00000009+315*1.007825032239+55*14.003074004439
+64*15.994914619579+6*31.97207117449 = **4727.1409**36279426

Molecular Weight #7: Net Charge

- The net formal charge on a molecule also affects the observed molecular weight due to electron mass.

```
int charge = 0;
for (ROMol::ConstAtomIterator atomIt = mol.beginAtoms();
     atomIt != mol.endAtoms(); ++atomIt)
  charge += (*atomIt)->getFormalCharge();

mw -= constants::electronMass * charge;
```

$$
\begin{aligned}
{}^{35}\mathrm{Cl}^{-} &= 34.968852682 + 0.00054857990943 \\
&= 34.969401262
\end{aligned}
$$

MOLECULAR WEIGHT #8: RELATIVITY

Using the rest mass of electrons neglects relativistic effects, observed as ionization energies (for cations) and electon affinities (for anions). $m = E/c^2$.

1st Ionization Energy of H = 1312.0 kJ/mol

Relativisitic Mass Correction = 0.000000014598

^{1}H = 1.00782503223 vs. ^{1}H^{+} = 1.00727646677

Electron Affinity of Cl = 348.575 kJ/mol

^{35}Cl^{-} = 34.968852682 + 0.00054857990943
+ 0.00000000387842 =
= 34.969401266

MOLECULAR WEIGHT #9: MAMW

The monoisotopic mass is not always the most abundant isotopologue of a compound, e.g. C_6Cl_6

C= 98.93% ^{12}C + 1.07% ^{13}C, Cl= 75.76% ^{35}Cl + 24.24% ^{37}Cl, Br= 50.69% ^{79}Br + 49.31% ^{81}Br

Carbon	**Chlorine**	**Bromine**
$1*0.9893^{6}$ = **93.75%**	$1*0.7576^{6}$ = 18.91%	$1*0.5069^{6}$ = 1.70%
$6*0.9893^{5}*0.0107$ = 6.08%	$6*0.7576^{5}*0.2424$ = **36.30%**	$6*0.5069^{5}*0.4931$ = 9.90%
$15*0.9893^{4}*0.0107^{2}$ = 0.16%	$15*0.7575^{4}*0.2424^{2}$ = 29.02%	$15*0.5069^{4}*0.4931^{2}$ = 24.08%
$20*0.9893^{3}*0.0107^{3}$ = 2.4E-3%	$20*0.7576^{3}*0.2424^{3}$ = 12.39%	$20*0.5069^{3}*0.4931^{3}$ = **31.23%**
$15*0.9893^{2}*0.0107^{4}$ = 1.9E-5%	$15*0.7576^{2}*0.2424^{4}$ = 2.07%	$15*0.5069^{2}*0.4931^{4}$ = 22.79%
$6*0.9893*0.0107^{5}$ = 8.33E-8%	$6*0.7576*0.2424{\wedge}^{5}$ = 0.38%	$6*0.5069*0.4931^{5}$ = 8.87%
$1*0.0107^{6}$ = 1.5E-10%	$1*0.2424^{6}$ = 2.0E-2%	$1*0.4931^{6}$ = 1.44%

C_6Cl_6 is typically $^{12}C_6{}^{35}Cl_5{}^{37}Cl$ and C_6Br_6 is $^{12}C_6{}^{79}Br_3{}^{81}Br_3$.

Efficient MAMW Calculation

- Proportion $P(n,k) = C(n,k)*p^{(n-k)}*(1-p)^k$
- Binomial coefficients $C(n,k) = n!/(k!(n-k)!)$ can be calculated incrementally, $C(n,k+1)=C(n,k)*(n-k)/(k+1)$
- The proportions drop when $(n-k)/(k+1)<(1-p)/p$.
- Therefore $(n+1)(1-p)-1 \leq k_{max} < (n+1)(1-p)$.

- Examples (n=6)
  - Carbon $p=0.9893$, $-0.9251 \leq k_{max} < 0.749$, $\therefore k_{max} = 0$
  - Chlorine $p=0.7576$, $0.6968 \leq k_{max} < 1.6968$, $\therefore k_{max} = 1$
  - Bromine $p=0.5069$, $2.4517 \leq k_{max} < 3.4517$, $\therefore k_{max} = 3$

MAMW WORKED EXAMPLE

Most abundant mass of Crambin ($C_{202}H_{315}N_{55}O_{64}S_6$)

- Carbon, n=202, p=0.9893, k_{max}=2
- Hydrogen, n=315, p=0.999815, k_{max}=0
- Nitrogen, n=55, p=0.99636, k_{max}=0
- Oxygen, n=64, p=0.99757, k_{max}=0
- Sulfur, n=6, p=0.9499, k_{max}=0

Top isotopologue: $^{12}C_{200}{}^{13}C_2{}^{1}H_{315}{}^{14}N_{55}{}^{16}O_{64}{}^{32}S_6$

lower = 200*12.00000000+2*13.003354835070+315*1.007825032230+55*14.003074004430+64*15.994914619570+6*31.97207117440 = **4729.1476**27765120

upper = 200*12.00000009+2*13.003354835079+315*1.007825032239+55*14.003074004439+64*15.994914619579+6*31.97207117449 = **4729.1476**45769584

MAMW is 4729.1476, monoisotopic was 4727.1409

END OF PART ONE

- If the quality/benefit of QSAR models depends upon the quality, accuracy and precision of input descriptors, then better calculations should (in theory) lead to better results.

LINE COUNT #1: BASELINE

- C++ (Python-like)

```
unsigned long result = 0;
std::ifstream ifs;
if (ifs.open(fname)) {
  std::string line;
  while (getline(ifs,line))
    result++;
  ifs.close();
}
return result;
```

- C (Old School)

```
unsigned long result = 0;
FILE *fp = fopen(fname,"r");
if (fp) {
  char buffer[65536];
  while (fgets(buffer,65536,fp))
    result++;
  fclose(fp);
}
return result;
```

LINE COUNT #2: CHAR AT A TIME

- The number of lines in a text file can be determined by counting the number of linefeed characters it contains.

```
unsigned long result = 0;
FILE *fp = fopen(fname,"r");
if (fp) {
  for (;;) {
    int ch = getc(fp);
    if (ch == '\n')
      result++;
    else if (ch == EOF)
      break;
  }
  fclose(fp);
}
return result;
```

LINE COUNT #3: GETC_UNLOCKED

- Opening the file in binary mode and using getc_unlocked can reduce processing per character.

```
unsigned long result = 0;
FILE *fp = fopen(fname,"rb");
if (fp) {
  for (;;) {
    int ch = getc_unlocked(fp);
    if (ch == '\n')
      result++;
    else if (ch == EOF)
      break;
  }
  fclose(fp);
}
return result;
```

Line Count #4: Reading Blocks

- Reading (and processing) blocks of data at a time, rather than characters is more efficient still.

```
char buf[BUFFER_SIZE+1];
int fd = open(fname,O_RDONLY|O_BINARY);
if (fd != -1) {
  fdadvise(fd,0,0,FADVISE_SEQUENTIAL);
  unsigned long result = 0;
  for (;;) {
    ssize_t count = read(fd,buff,BUFFER_SIZE);
    if (count > 0)
      result += process_buffer(buff,count);
    } else break;
  }
  close(fd);
}
return result;
```

```
static inline unsigned long
process_buffer(const char *ptr,
               unsigned int count)
{
  const char *end = ptr+count;
  unsigned long result = 0;
  while (ptr < end) {
    if (*ptr == ‘\n’)
      result++;
    ptr++;
  }
  return result;
}
```

LINE COUNT #5: UNIX MMAP

- More efficient still is the use of memory mapping.

```
int fd = open(fname,O_RDONLY);
unsigned long result = 0;
if (fd != -1) {
  struct stat buf;
  if (fstat(fd,&buf) >= 0 && S_ISREG(buf.st_mode)) {
    unsigned long len = (unsigned long)buf.st_size;
    if (len > 0) {
      fdadvise(fd,0,0,FADVISE_SEQUENTIAL);
      char *ptr = (char*)mmap(0,len,PROT_READ,MAP_FILE|MAP_PRIVATE,fd,0);
      if (ptr != MAP_FAILED) {
        result = process_buffer(ptr,len);
        munmap(ptr,len);
      }
    }
  }
}
close(fd);
return result;
```

LINE COUNT #6: UNROLLING

- Avoiding the if-statement in the process_block loop is a big win, but modern compiler automatically unroll and/or reverse the small loops if that helps.

```
unsigned int
process_block(const char *ptr,
              unsigned int n)
{
  unsigned int result = 0;
  for (unsigned int i=0; i<n; i++)
    result += (ptr[i] == '\n’);
  return result;
}
```

```
unsigned int result = 0;
do {
  if (*ptr++==‘\n’) result++;
  if (*ptr++==‘\n’) result++;
  if (*ptr++==‘\n’) result++;
  if (*ptr++==‘\n’) result++;
  if (*ptr++==‘\n’) result++;
  if (*ptr++==‘\n’) result++;
  if (*ptr++==‘\n’) result++;
  if (*ptr++==‘\n’) result++;
  n -= 8;
} while (n);
return result;
```

Line count #7: memchr

- The Linux utility wc uses the standard C libraries memchr function to efficiently find the next ‘\n’.

```
unsigned int process_block(const char *ptr, unsigned int n) {
  const char *end = ptr+n;
  unsigned int result = 0;
  while ((ptr = (const char*)memchr(ptr,'\n',end-ptr))) {
    result++;
    ptr++;
  }
  return result;
}
```

LINE COUNT #8: BIT TWIDDLING

- It’s possible, if the buffer is suitably aligned, to process multiple characters at a time using “bit twiddling hacks”.

```
unsigned int process_block(const char *buffer, unsigned int n) {
  unsigned long long *ptr = (unsigned long long*)buffer;
  unsigned int result = 0;
  do {
    unsigned long long temp = *ptr++;
    temp ^= 0x0a0a0a0a0a0a0a0all;
    temp |= ((temp & 0x7f7f7f7f7f7f7f7fll) + 0x7f7f7f7f7f7f7f7fll);
    temp = temp | 0x7f7f7f7f7f7f7f7fll;
    if (temp != ~(long long)0)
      result += __builtin_popcountll(~temp);
    n -= 8;
  } while (n);
  return result;
}
```

Line Count #9: SIMD Instructions

- On Intel chips, we can use (AVX) vector instructions.

```
unsigned int process_block(const char *buffer, unsigned int n) {
const __m128i *ptr = (const __m128i*)buffer;
const __m128i c = _mm_set1_epi8('\n');
unsigned int result = 0;
do {
  unsigned int mask1 = _mm_movemask_epi8(_mm_cmpeq_epi8(*ptr++,c));
  unsigned int mask2 = _mm_movemask_epi8(_mm_cmpeq_epi8(*ptr++,c));
  unsigned int mask3 = _mm_movemask_epi8(_mm_cmpeq_epi8(*ptr++,c));
  unsigned int mask4 = _mm_movemask_epi8(_mm_cmpeq_epi8(*ptr++,c));
  result += _mm_popcnt_u64((unsigned long)mask1 +
                           (((unsigned long)mask2)<<16) +
                           (((unsigned long)mask3)<<32) +
                           (((unsigned long)mask4)<<48));
  n -= 64;
} while(n);
return result;
```

QUESTIONING THE RULES

- "Never read a file backwards" – Rosemary Francis in "Best practices when accessing Big Data or any other data!", ACCU Cambridge, 14th March 2018.
- Consider the challenge of processing a file larger than memory: each sequential scan of the file defaults the LRU caching of the operating system.
- Processing records already in cache/memory first improves efficiency.
- This is one reason relational databases/SQL don't (by default) guarantee the ordering of result rows.

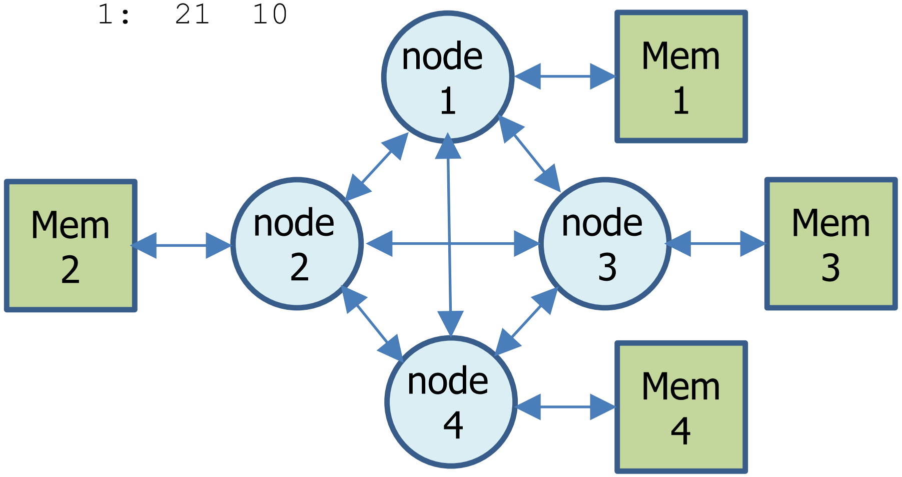
LINE COUNT #10: MINCORE

```
unsigned long pagesize, pages, i;
unsigned char *vec, *beg;

pagesize = sysconf(_SC_PAGESIZE);
pages = (len+pagesize-1)/pagesize;

vec = (unsigned char*)malloc(pages);
if (vec) {
  if (mincore(ptr,len,vec) == 0) {
    unsigned long incore = 0;

    /* first pass: in memory */
    for (i=0; i<pages; i++) {
      if ((vec[i]&1) != 0) {
        beg = ptr+(i*pagesize);
        result+=process_block(beg,
                              pagesize);
        incore++;
      }
    }
    /* second pass: on disk */
    if (incore == 0)
      result = process_block(ptr,len);
    else if (incore != pages) {
      for (i=0; i<pages; i++) {
        if ((vec[i]&1) == 0) {
          beg = ptr+(i*pagesize);
          result += process_block(beg,
                                  pagesize);
        }
      }
    }
  } else
    result = process_block(ptr,len);
  free(vec);
} else
  result = process_block(ptr,len);
```

c.f. Linux utility vmtouch

NUMA MEMORY ARCHITECTURE

```
mother% numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 8 9 10 11
node 0 size: 65403 MB
node 0 free: 4469 MB
node 1 cpus: 4 5 6 7 12 13 14 15
node 1 size: 65536 MB
node 1 free: 54 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```



```
smallworld% numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1
node 0 size: 65533 MB
node 0 free: 1218 MB
...
node 7 cpus: 14 15
node 7 size: 65520 MB
node 7 free: 17354 MB
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  16  16  22  16  22  16  22
  1:  16  10  22  16  22  16  22  16
  2:  16  22  10  16  16  22  16  22
  3:  22  16  16  10  22  16  22  16
  4:  16  22  16  22  10  16  16  22
  5:  22  16  22  16  16  10  22  16
  6:  16  22  16  22  16  22  10  16
  7:  22  16  22  16  22  16  16  10
```

Influence on Performance

```
linux% ls -hs pubchem.smi
3.3G pubchem.smi

linux% ./numatouch pubchem.smi
861588 pages of 4096 bytes
861588 pages are in memory
  614832 pages on NUMA node 1
  448 pages on NUMA node 2
  5792 pages on NUMA node 3
  240516 pages on NUMA node 7
```

```
linux% time /usr/bin/wc -l pubchem.smi
53923723 pubchem.smi

real    0m2.158s
user    0m0.859s
sys     0m1.296s
linux% time /usr/bin/wc -l pubchem.smi
53923723 pubchem.smi

real    0m1.844s
user    0m0.902s
sys     0m0.940s
linux% time /usr/bin/wc -l pubchem.smi
53923723 pubchem.smi

real    0m2.625s
user    0m0.904s
sys     0m1.718s
```

LINE COUNT #11: CPU AFFINITY

```
#define MPOL_F_NODE 1
#define MPOL_F_ADDR 2
int addr2node(void *addr) {
  int node = -1;
  syscall(__NR_get_mempolicy, &node,
          0, 0, addr,
          MPOL_F_NODE|MOL_F_ADDR);
  return node;
}

unsigned long long node_pages[64];
memset(node_psges,0,sizeof(node_pages));

for (i=0; i<pages; i++) {
  if ((vec[i]&1) != 0) {
    beg = ptr+(i*pagesize);
    int node = addr2node(beg);
    if (node>=0 && node<64)
      node_pages[node]++;
  }
}
```

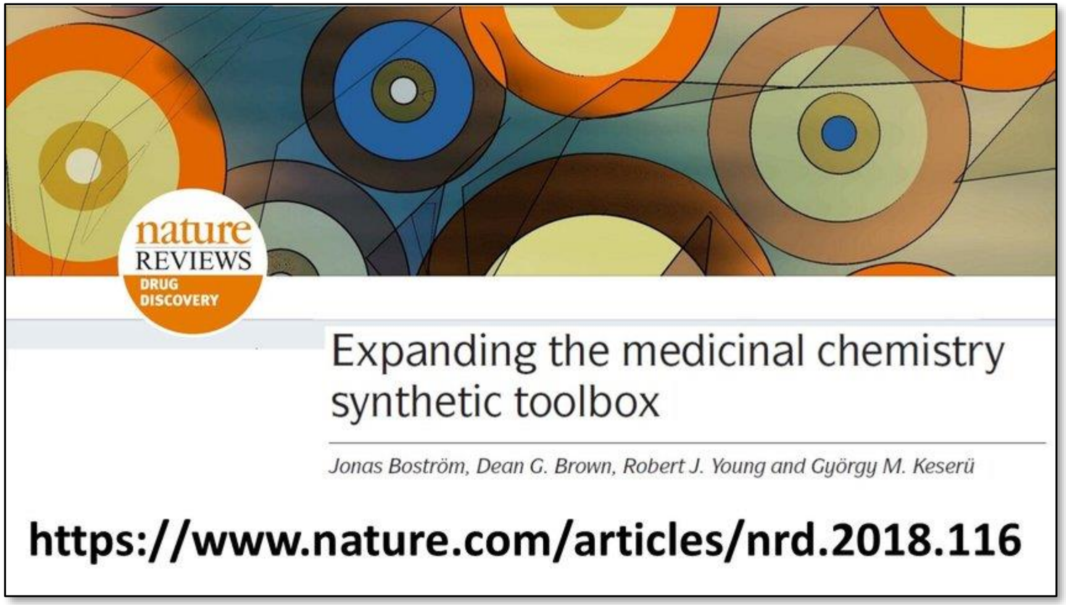
```
unsigned int best = 0;
for (unsigned int j=1; j<64; j++)
  if (node_pages[j] > node_pages[best])
    best = j;
if (node_pages[best]) {
  cpu_set_t set;
  if (numa_node_to_cpuset(best,&set))
    sched_setaffinity(0,sizeof(set),&set);
}
```

END OF PART TWO

- Optimally counting the number of lines in a text file can require low-level programming.
- In practice, (cheminformatics) programs that split their work over multiple threads, need to coordinate the CPU cores that those threads execute on, with the NUMA nodes that hold (the fraction of the) database being processed.

OUTRAGEOUS CLAIM OF THE WEEK

- "Five major reactions are … most frequently used and represent more than 80% of reactions used for drug discovery."



nature REVIEWS DRUG DISCOVERY

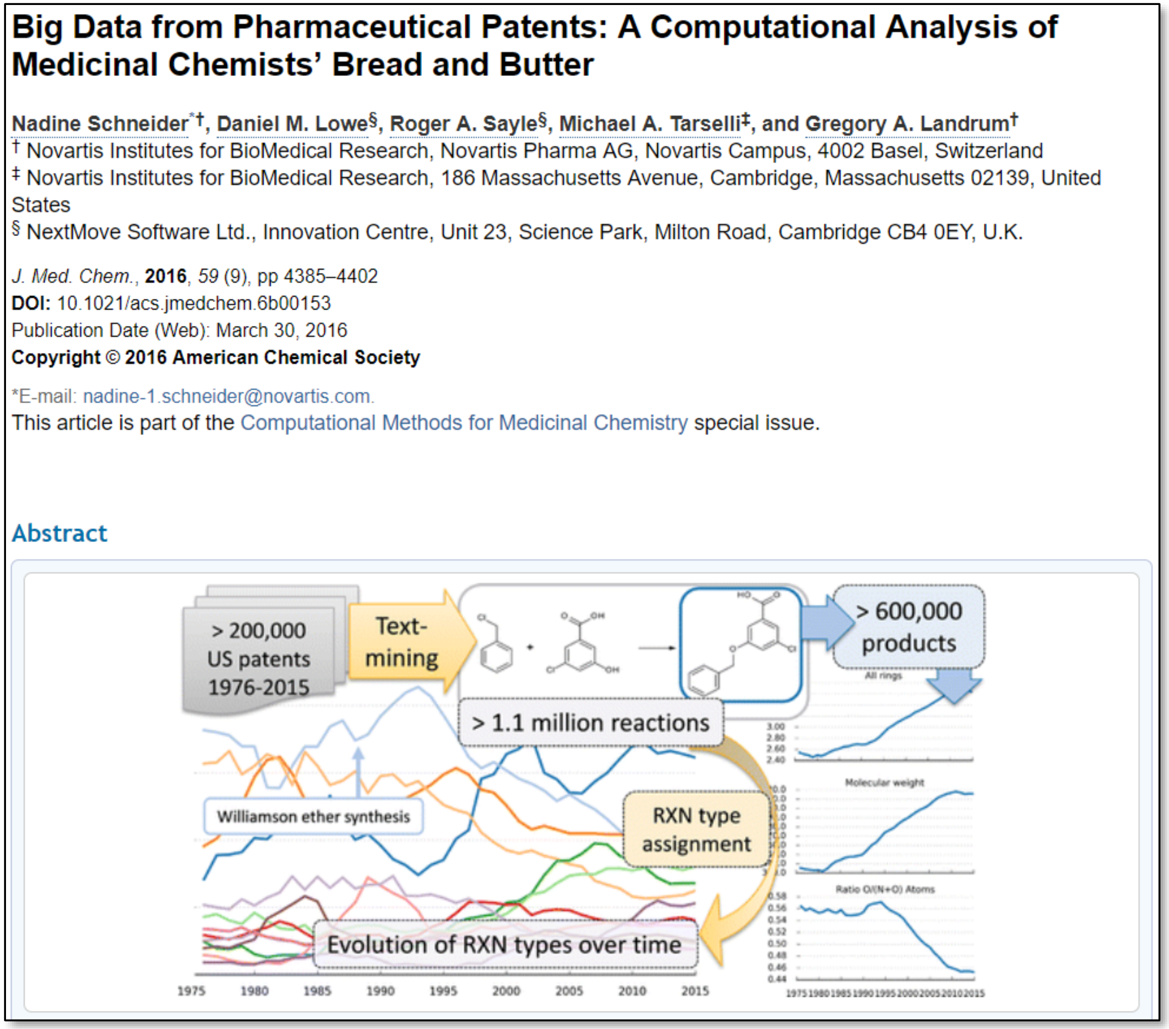
Expanding the medicinal chemistry synthetic toolbox

Jonas Boström, Dean G. Brown, Robert J. Young and György M. Keserü

https://www.nature.com/articles/nrd.2018.116

https://t.co/SlUwPtknQi

THAT DOESN'T SOUND RIGHT



Big Data from Pharmaceutical Patents: A Computational Analysis of Medicinal Chemists' Bread and Butter

Nadine Schneider*†, Daniel M. Lowe§, Roger A. Sayle§, Michael A. Tarselli‡, and Gregory A. Landrum†

† Novartis Institutes for BioMedical Research, Novartis Pharma AG, Novartis Campus, 4002 Basel, Switzerland

‡ Novartis Institutes for BioMedical Research, 186 Massachusetts Avenue, Cambridge, Massachusetts 02139, United States

§ NextMove Software Ltd., Innovation Centre, Unit 23, Science Park, Milton Road, Cambridge CB4 0EY, U.K.

J. Med. Chem., **2016**, *59* (9), pp 4385–4402

DOI: 10.1021/acs.jmedchem.6b00153

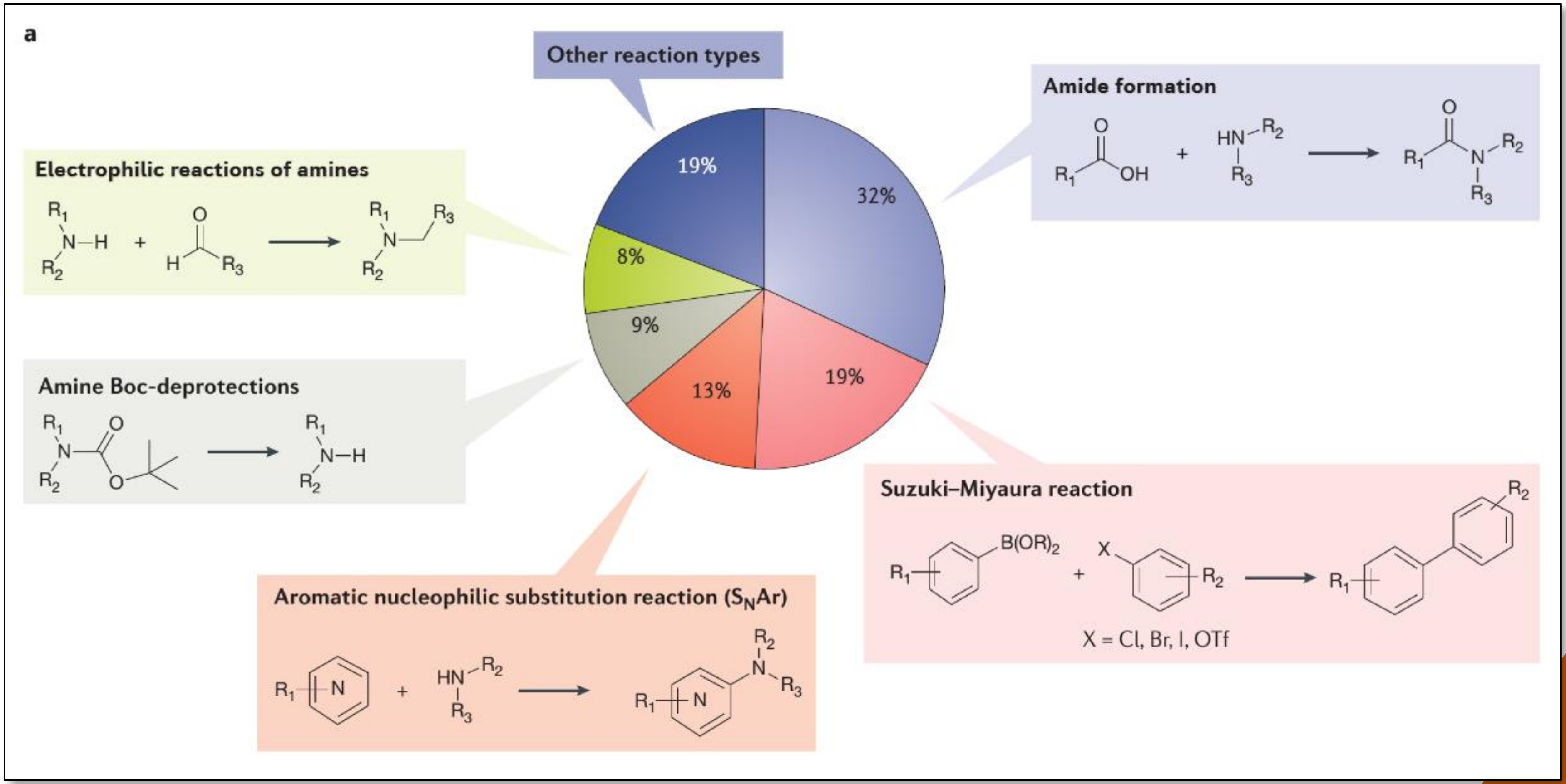
Publication Date (Web): March 30, 2016

*E-mail: nadine-1.schneider@novartis.com.

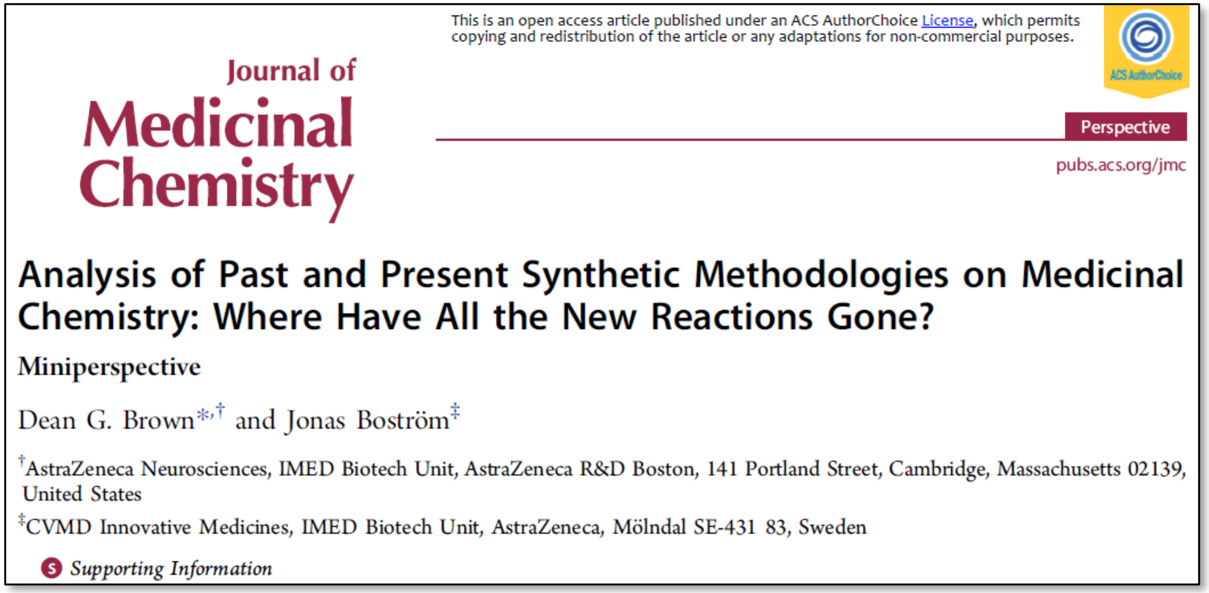
This article is part of the Computational Methods for Medicinal Chemistry special issue.

Abstract

HERE'S THE PIE CHART



SHOW ME THE DATA



Journal of Medicinal Chemistry

Perspective

pubs.acs.org/jmc

Analysis of Past and Present Synthetic Methodologies on Medicinal Chemistry: Where Have All the New Reactions Gone?

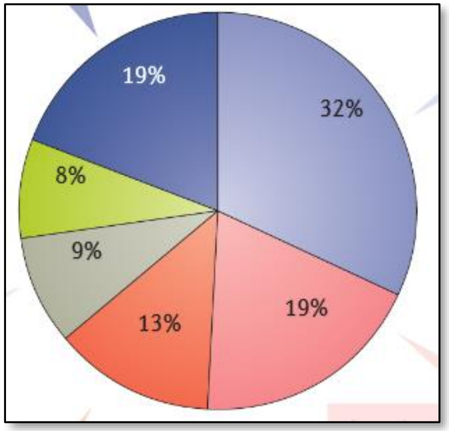
Miniperspective

Dean G. Brown*[†] and Jonas Boström[‡]

[†]AstraZeneca Neurosciences, IMED Biotech Unit, AstraZeneca R&D Boston, 141 Portland Street, Cambridge, Massachusetts 02139, United States

[‡]CVMD Innovative Medicines, IMED Biotech Unit, AstraZeneca, Mölndal SE-431 83, Sweden

Supporting Information



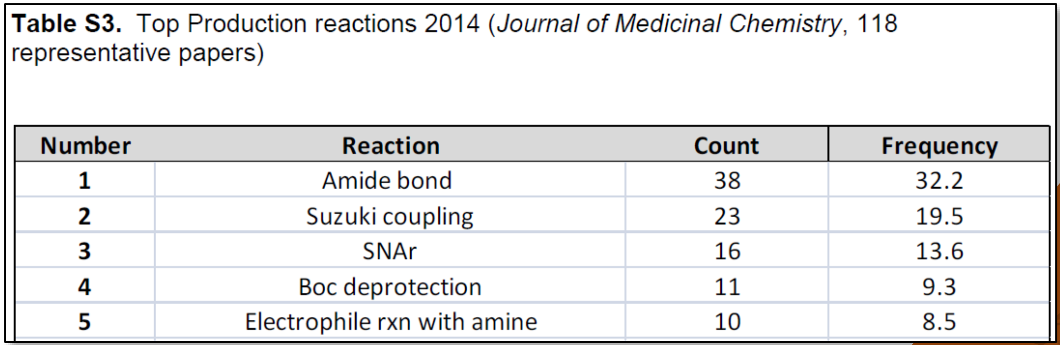


Table S3. Top Production reactions 2014 (*Journal of Medicinal Chemistry*, 118 representative papers)

Number	Reaction	Count	Frequency
1	Amide bond	38	32.2
2	Suzuki coupling	23	19.5
3	SNAr	16	13.6
4	Boc deprotection	11	9.3
5	Electrophile rxn with amine	10	8.5

SHOULDN'T PERCENTAGES SUM TO 100?

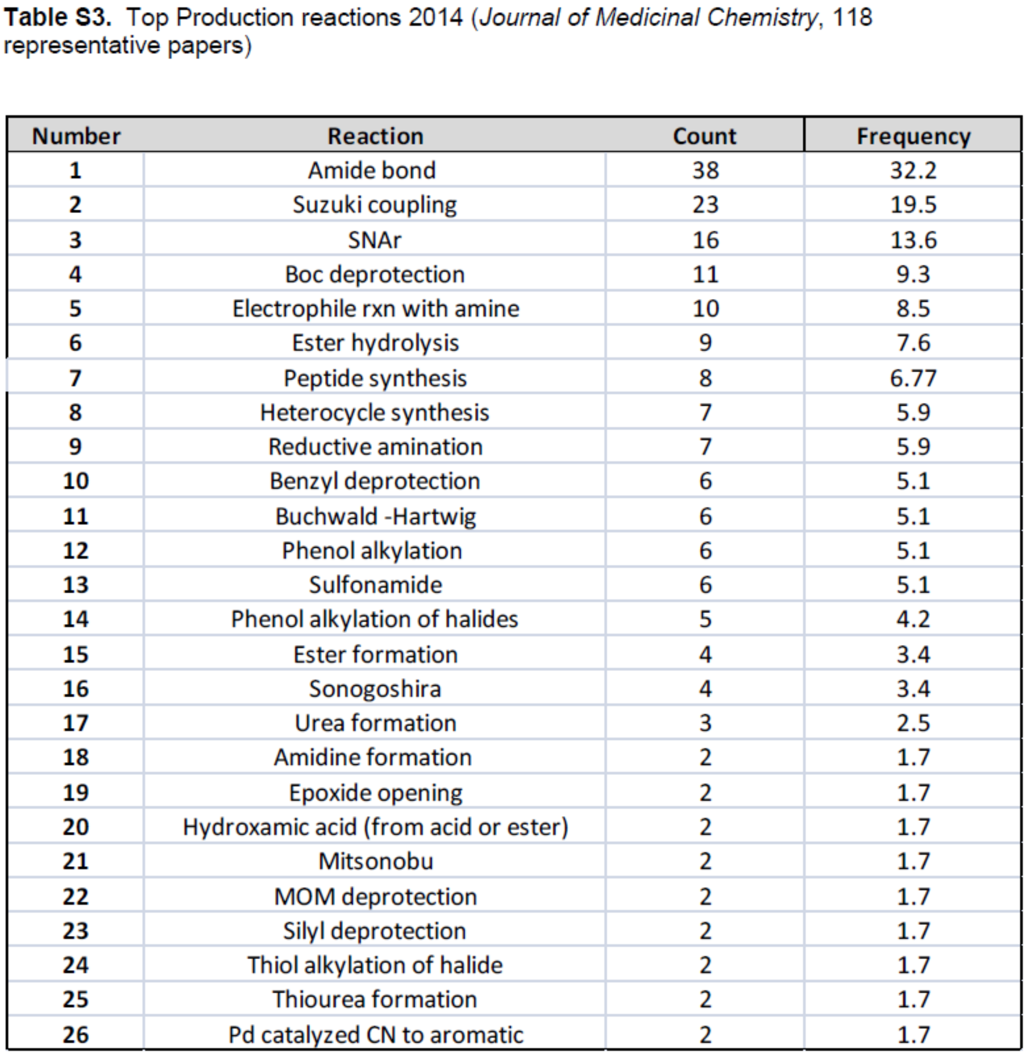


Table S3. Top Production reactions 2014 (*Journal of Medicinal Chemistry*, 118 representative papers)

Number	Reaction	Count	Frequency
1	Amide bond	38	32.2
2	Suzuki coupling	23	19.5
3	SNAr	16	13.6
4	Boc deprotection	11	9.3
5	Electrophile rxn with amine	10	8.5
6	Ester hydrolysis	9	7.6
7	Peptide synthesis	8	6.77
8	Heterocycle synthesis	7	5.9
9	Reductive amination	7	5.9
10	Benzyl deprotection	6	5.1
11	Buchwald -Hartwig	6	5.1
12	Phenol alkylation	6	5.1
13	Sulfonamide	6	5.1
14	Phenol alkylation of halides	5	4.2
15	Ester formation	4	3.4
16	Sonogoshira	4	3.4
17	Urea formation	3	2.5
18	Amidine formation	2	1.7
19	Epoxide opening	2	1.7
20	Hydroxamic acid (from acid or ester)	2	1.7
21	Mitsonobu	2	1.7
22	MOM deprotection	2	1.7
23	Silyl deprotection	2	1.7
24	Thiol alkylation of halide	2	1.7
25	Thiourea formation	2	1.7
26	Pd catalyzed CN to aromatic	2	1.7

Σcount = 187
Σfrequency=158.47
frequency=count/papers

1. 20.3%
2. 12.3%
3. 8.6%
4. 5.9%
5. 5.3%

Top 5 rxn types (from a sample of 187) account for ~52.4% of rxns.

MATTERS OF OPINION

- I’d question whether we actually need to extend the medicinal chemist’s toolbox.
- “Everything should be made as simple as possible, but not simpler” – Albert Einstein.
- Examples: Ink jet printers, jet fighter maintenance, RISC CPUs, virtual machines, compilers…
- Perhaps what’s wanted is the smallest number of reactions that supports the broadest range of chemistry, i.e. shrink (replace?) the toolbox.

Acknowledgements

- Many thanks to Greg Landrum for putting up with me year after year.
- Many thanks to my colleagues at NextMove Software who put up with me day in, day out.
- Thomas Blaschke and Andrew Dalke.
- Thank you for your time.