CHEMINFORMATICS @ UNIVERSITI SAINS MALAYSIA

NURUL HASHIMAH AHAMED HASSAIN MALIM

PRAGMA 33 MEETING, BRISBANE

17 OKTOBER 2017

PAST & PRESENT

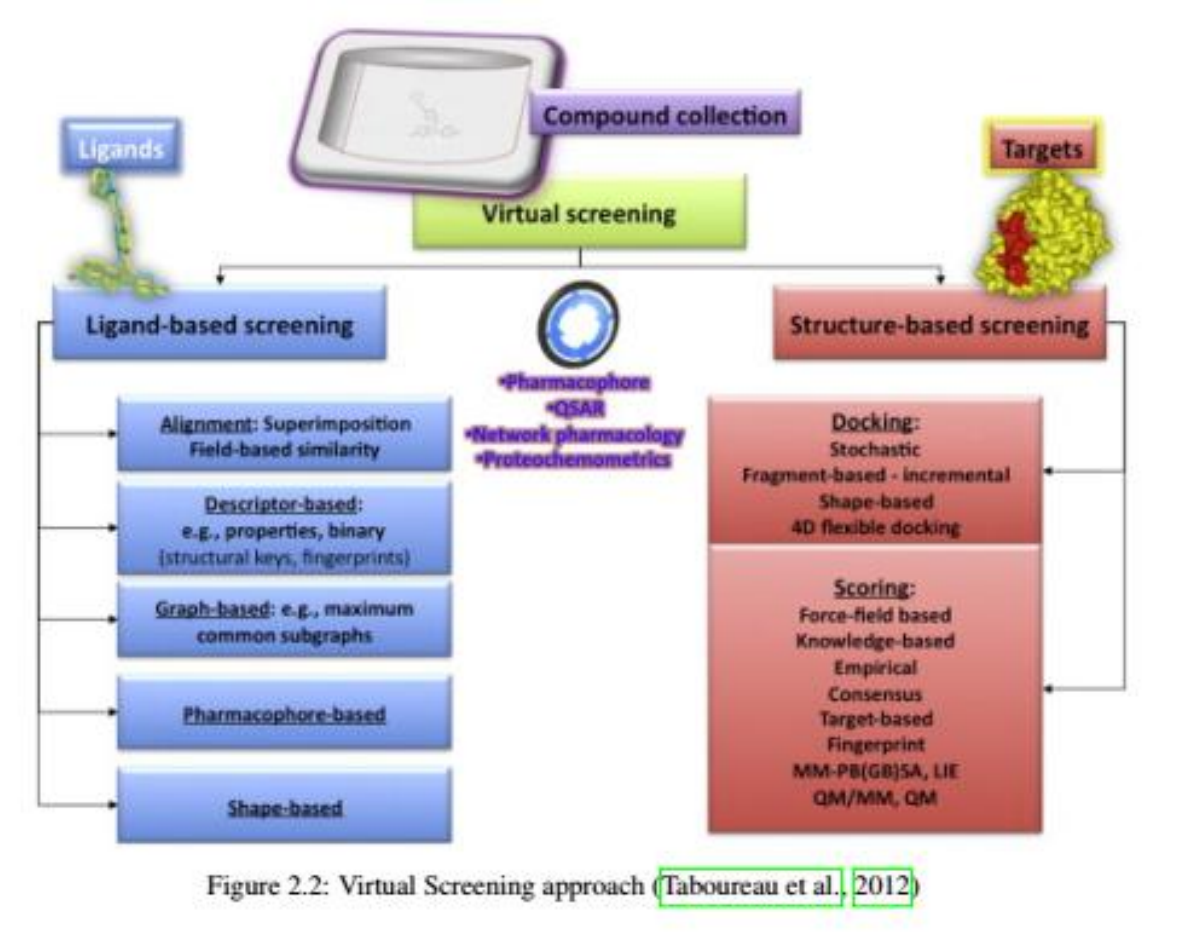
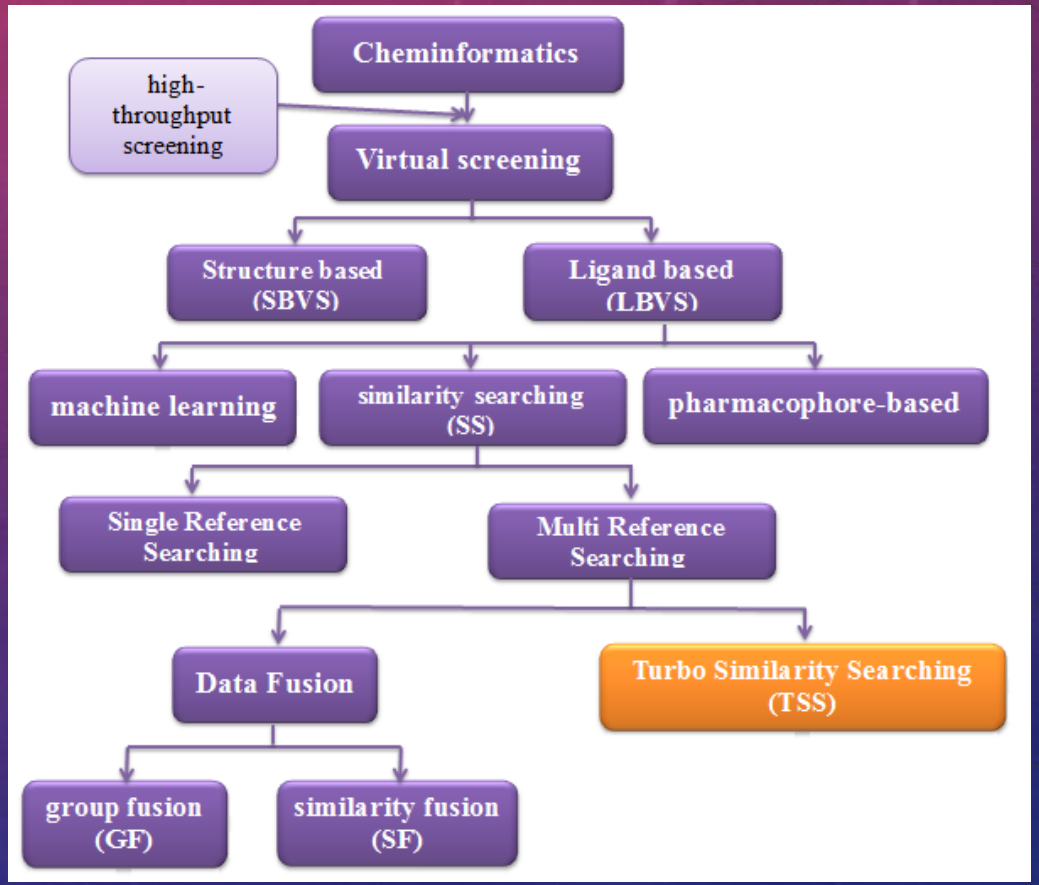


Figure 2.2: Virtual Screening approach (Taboureau et al., 2012)



SIMILARITY SEARCHING – AN ILLUSTRATION

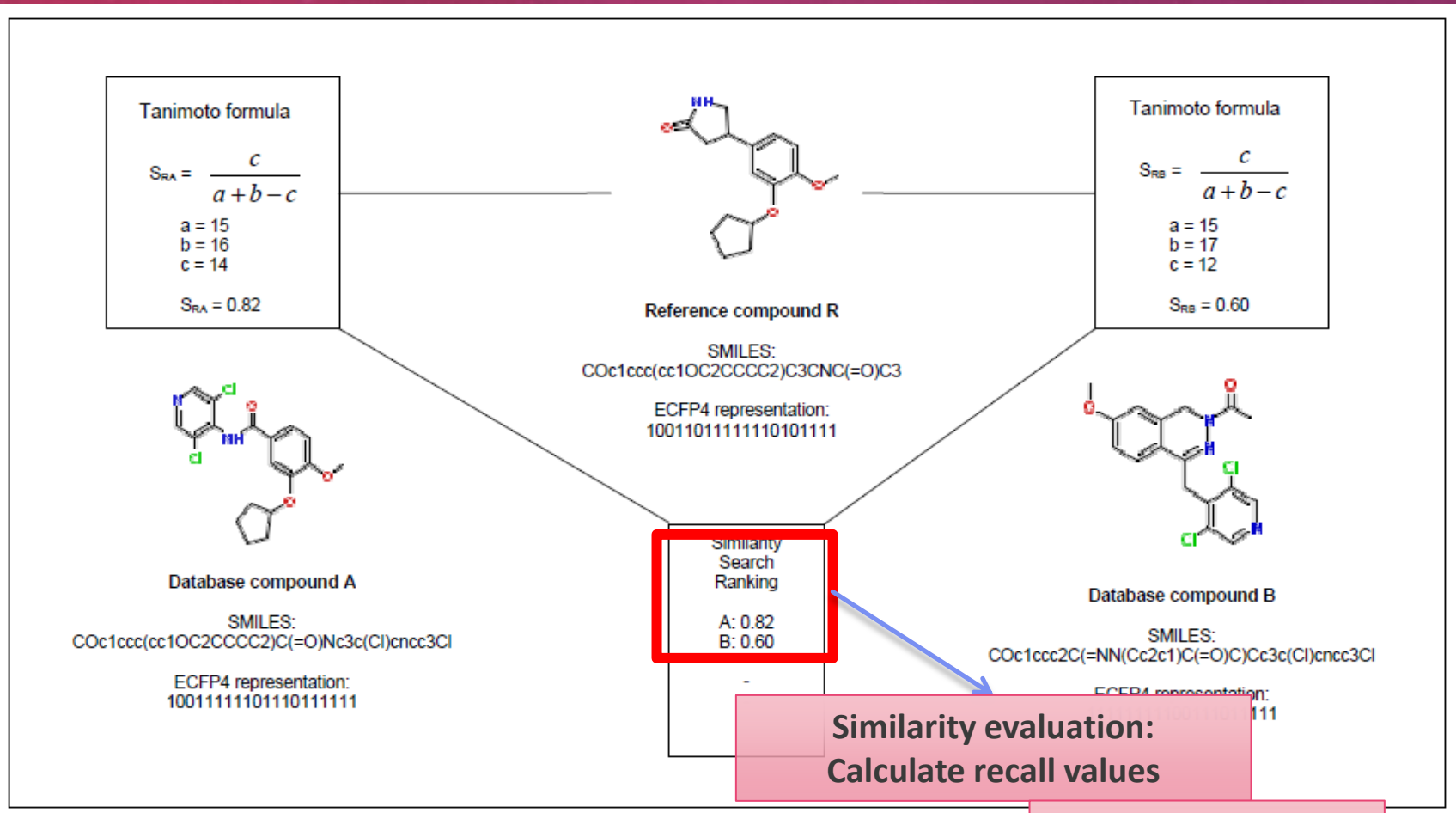
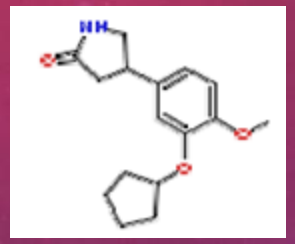


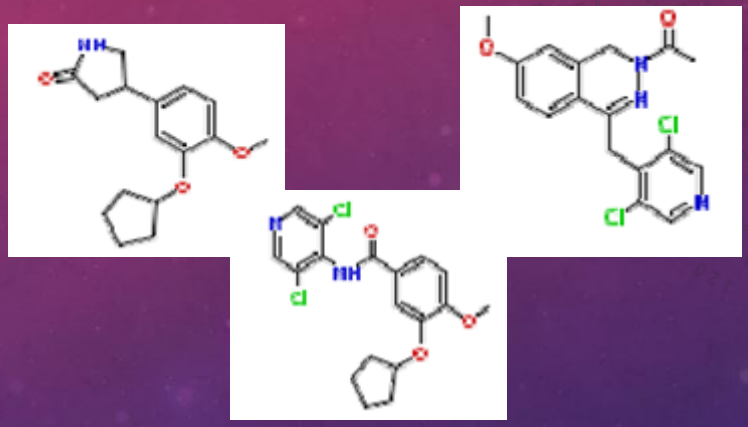
Figure 2.1: Illustration of similarity searching using ECFP4 and Tanimoto

THE TRANSITION – ENHANCING SIMILARITY SEARCHING

One target search



Multiple target search



Similar Property Principle

Neighbourhood Behaviour

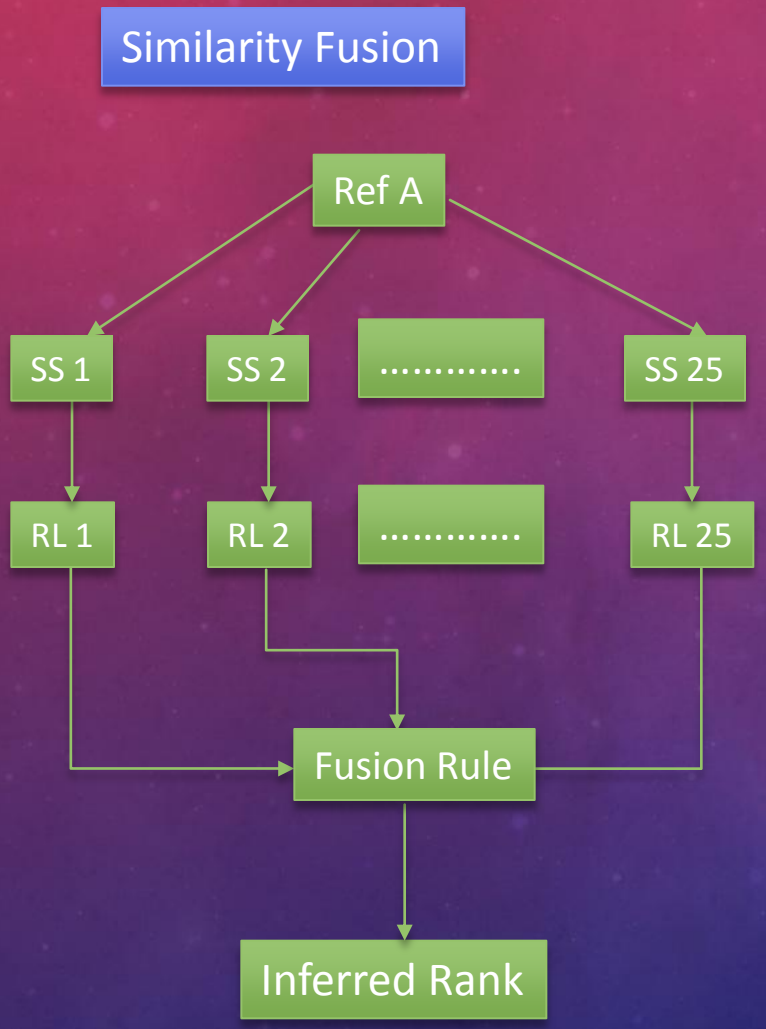
Direct Relationship (Target-Database)

Indirect Relationship (Target-Database)

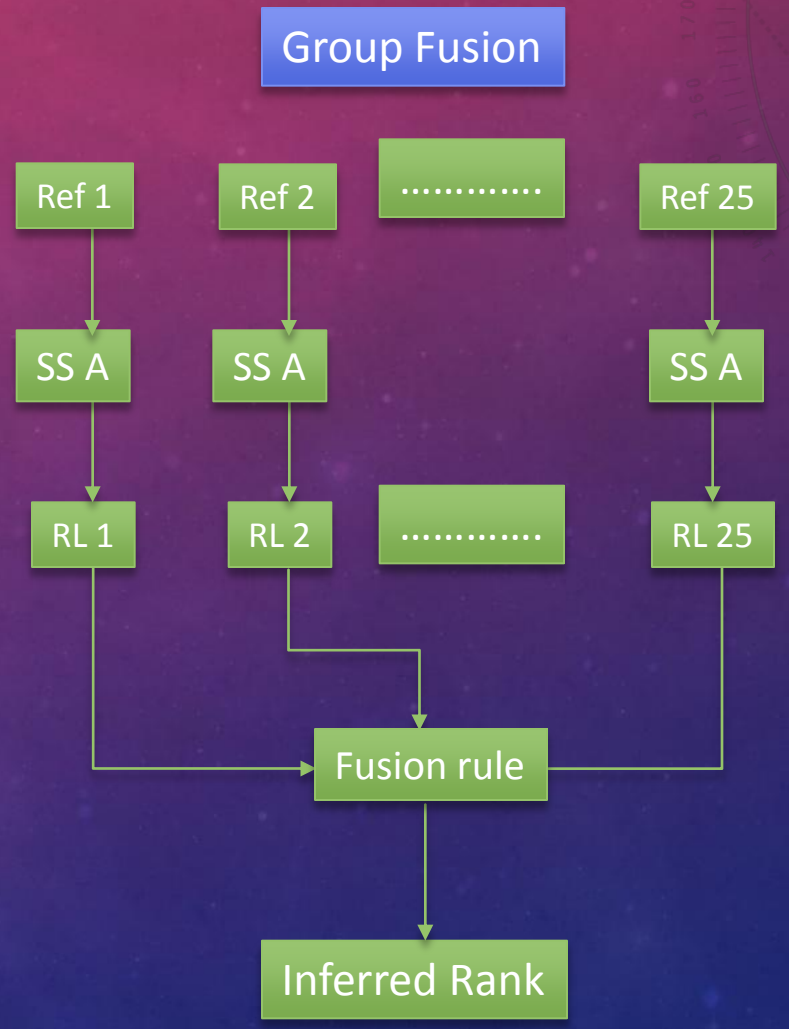
Similarity Search

Data Fusion; Turbo Similarity Search

TURBO SIMILARITY SEARCHING (WILLETT'S GROUP, SHEFFIELD)



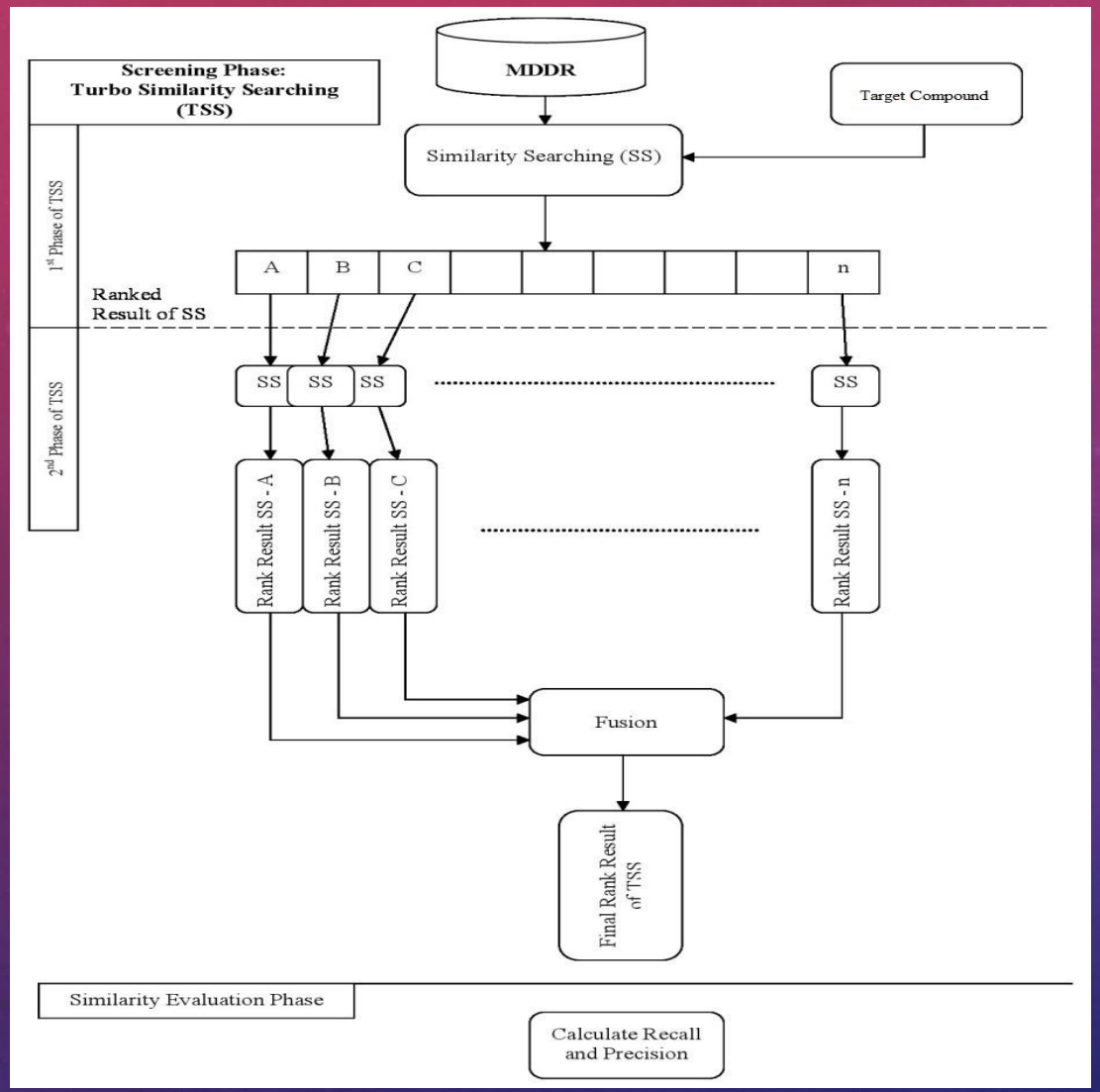
Similarity measures:
Fingerprint: MDL, Unity, ECFP4, BCI and Daylight
Coefficient: Tanimoto, Cosine, Forbes, Simple Match, Russell-Rao



Similarity measures:
Fingerprint: ECFP4
Coefficient: Tanimoto

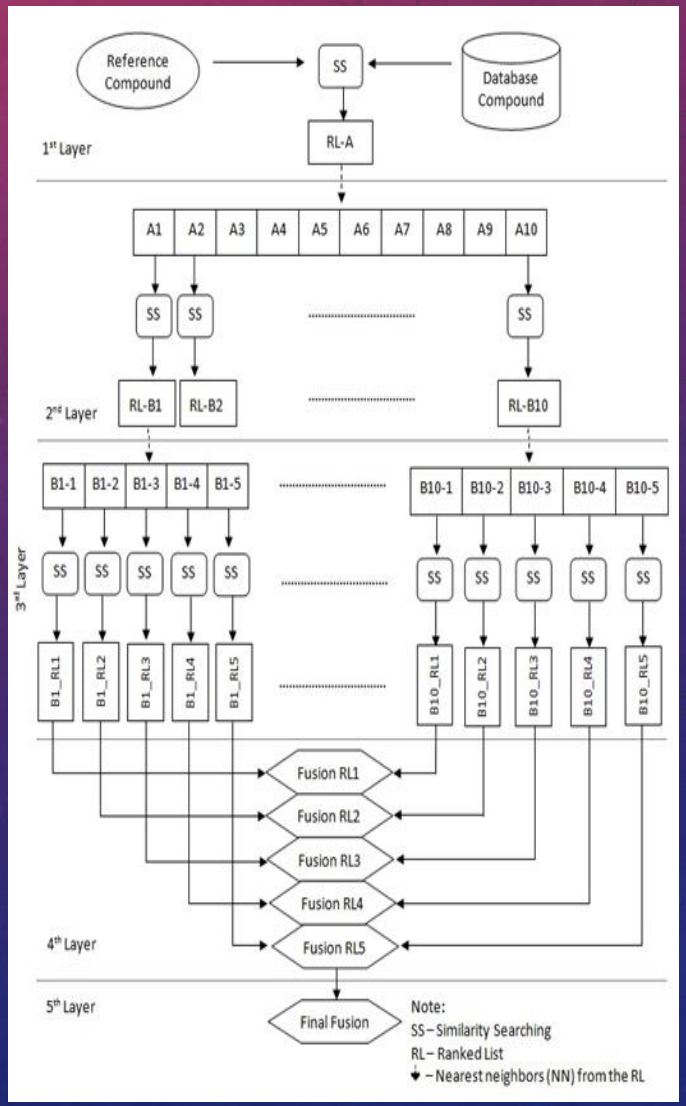
TURBO SIMILARITY SEARCHING (WILLETT'S GROUP, SHEFFIELD)

+ ALIA AZLEEN, NORASYIKIN YUSRI, YONG PEI CHIA, ABEER ALHASBARY, NURUL HASHIMAH AHAMED HASSAIN MALIM



Multi-level of indirect relationship

Fusion Rules, Similarity Measures Combination



SIMILARITY FUSION ON OPENMP & CUDA

MOSTAFA ALBARMAWI, NURUL HASHIMAH AHAMED HASSAIN MALIM

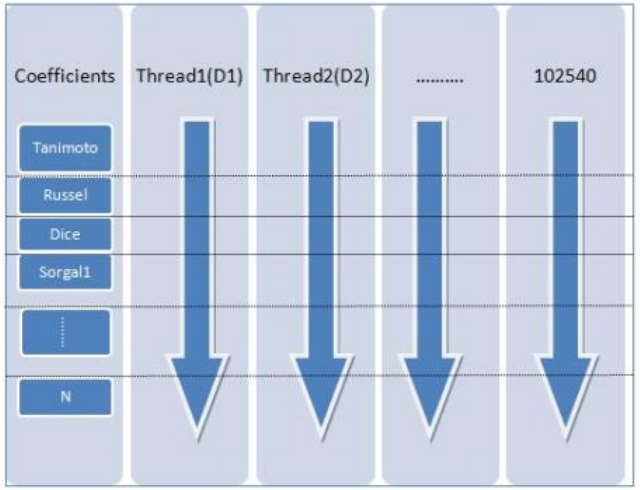


Figure 3.8: Chemical similarity computation decomposed on GPU

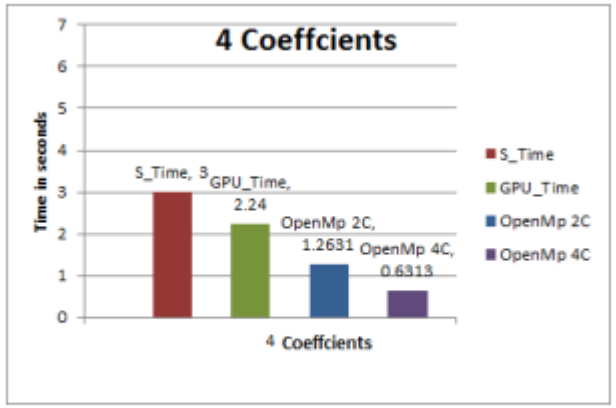


Figure 4.3: The Execution of Four Coefficients in Different Platforms

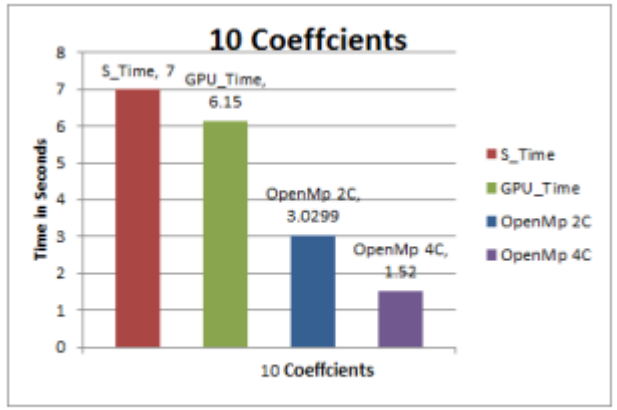
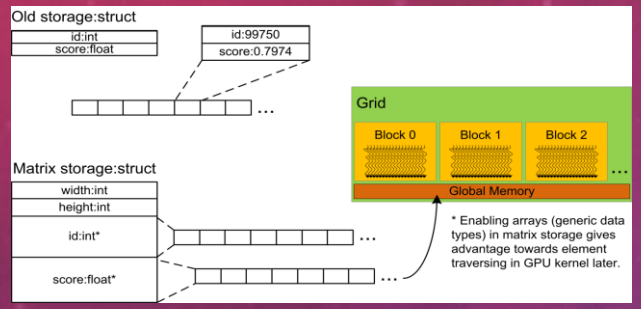
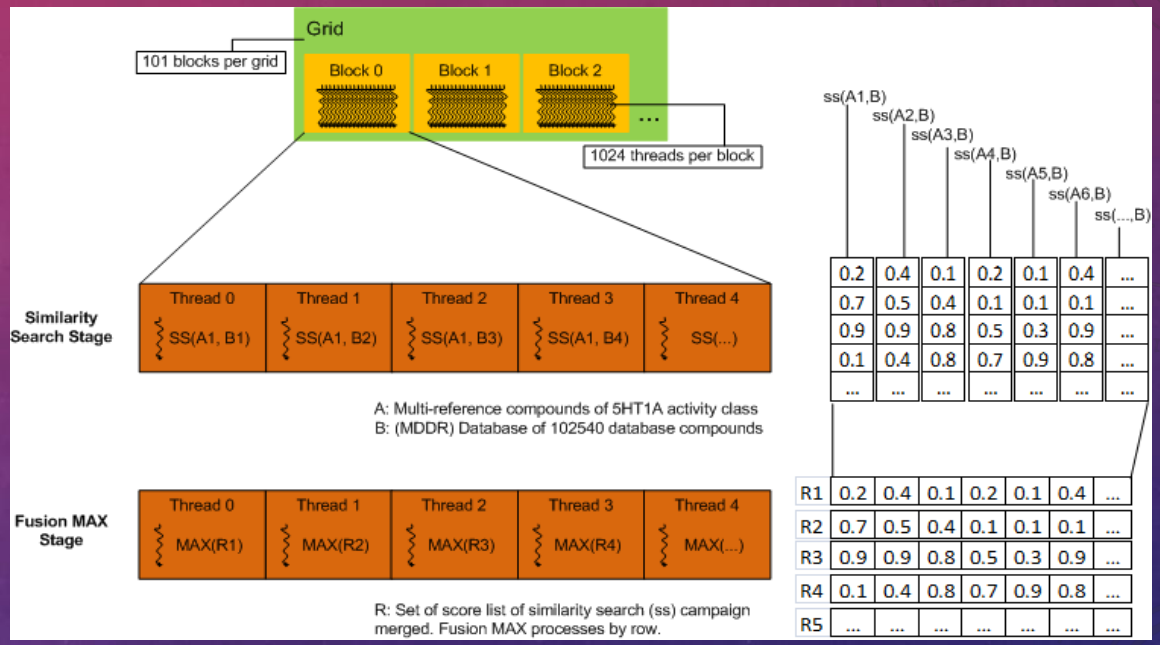


Figure 4.9: The Execution of Ten Coefficients in Different Platforms

GROUP FUSION ON OPENMP & CUDA

MOHD NORHADRI HILMI, NURUL HASHIMAH AHAMED HASSAIN MALIM





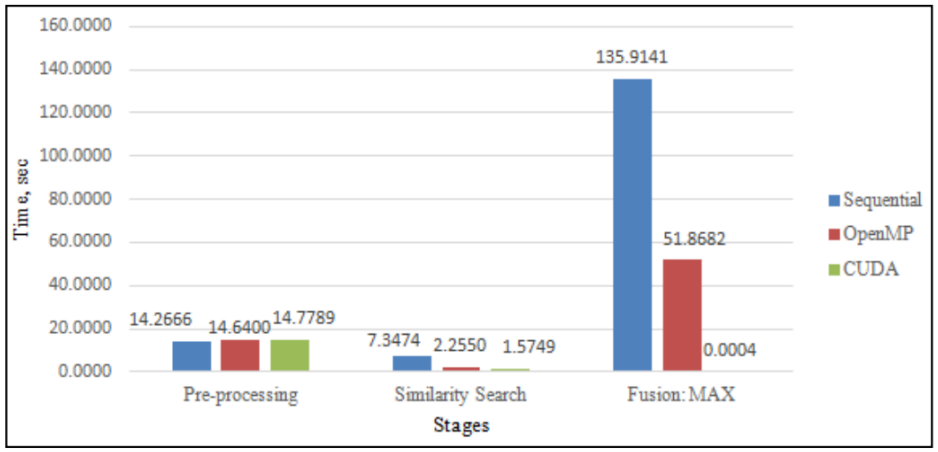


Fig. 9. Comparison on Sequential, parallel OpenMP and parallel CUDA for 10 query compounds.

TSS ON CUDA & OPENMP

MARWAH HAITHAM AL-LAILA, NURUL HASHIMAH AHAMED HASSAIN MALIM

```
//SS for the 2nd phase
void SS2(TanVal *A,int rowa, tScreen *DB,int rowd,TanVal *Rank)
{
   int i,j,c=0;
   float h=0;

   #pragma omp parallel for private(c,h,j) num_threads(no_proc)
    for(i=0; i<rowd; i++)
       {
              Rank[i].id=DB[i].id;
              c=0;
              for (j=0; j<NumBits; j++)
                  {   if (( A[rowa].screen[j]==1)&&(DB[i].screen[j]==1))
                         c++;
                      Rank[i].screen[j]=DB[i].screen[j];
                  }
              h=(A[rowa].bitsset+DB[i].bitsset-c);
              Rank[i].bitsset=DB[i].bitsset;
              Rank[i].value=c/h;
       }

}

//GF function for the 2nd phase
void Fusion(TanVal *R1,TanVal *R2)
{
       int i,j;

       #pragma omp parallel for num_threads(no_proc)
       for (i=0; i<wombat; i++)
         {
              if (R2[i].value>R1[i].value)
                {
                       R1[i].id=R2[i].id;
                       R1[i].value=R2[i].value;
                       R1[i].bitsset=R2[i].bitsset;
                         for (j=0; j<NumBits; j++)
                                  R1[i].screen[j]=R2[i].screen[j];
                }
         }
}
```

Figure 3.13: Pragma directives on SS and GF

Procedure TSS(R, DB[0: n-1])
Input: R (one structure, reference structure)
DB[0:n-1]) (an array of structure with *n* element size, database)
Output: L_i [0:n-1] (i number of List L, L is array with *n* element size)

Allocate the memory on the CPU for (R, DB, List) with size (1, n, n)
Read (R, DB)
No_threads ←1024 ; //maximum no. of thread in block
If n% No_threads == 0 **then** //n is even
No_blocks ← n/ No_threads;
else No_blocks ← n/ No_threads +1; //n is odd
endif
cudaMalloc(R_d, 1)
cudaMalloc(DB_d, n)
cudaMalloc(List_d, n)
cudaMemcpy(R_d, R, 1, cudaMemcpyHostToDevice);
cudaMemcpy(DB_d, DB, n, cudaMemcpyHostToDevice);
cudaMemcpy(List2_d, List, 200, cudaMemcpyHostToDevice);

SS1_kernel <<<No_blocks, No_threads >>> (R_d,DB_d,n,List0_d);
cudaMemcpy(List, List_d, n,cudaMemcpyDeviceToHost);

QuickSort(List)
cudaMemcpy(List2_d List, 200, cudaMemcpyHostToDevice);
cudaMemcpy(L_d L, (200*n), cudaMemcpyHostToDevice);
SS2_Fusion_kernel <<< No_blocks, No_threads >>> (List2_d,200, DB_d,n, L_d,n,L)
cudaDeviceSynchronize();

cudaMemcpy(L, L_d,(9*n),cudaMemcpyDeviceToHost);
write (L) to text files
end TSS

Figure 3.6: Pseudocode of the Parallel TSS algorithm using CUDA method 1

Procedure TSS (R, DB[0:n-1])
Input: R (one structure, reference structure)
DB[0:n-1]) (an array of structure with *n* element size, database)
Output: L_i [0:n-1] (i number of List L, L is array with *n* element size)
Allocate the memory on the CPU for (R, DB, List) with size (1, n, n)
Read (R,DB)

No_threads ←1024 ; //maximum no. of thread in block
If n% No_threads == 0 **then** //n is even
No_blocks ← n/ No_threads;
else No_blocks ← n/ No_threads +1; //n is odd
endif
cudaMalloc(R_d, 1)
cudaMalloc(DB_d, n)
cudaMalloc(List_d, n)
cudaMemcpy(R_d, R, 1, cudaMemcpyHostToDevice);
cudaMemcpy(DB_d, DB, n, cudaMemcpyHostToDevice);
cudaMemcpy(List2_d, List, 200, cudaMemcpyHostToDevice);
SS1_kernel <<<No_blocks, No_threads >>> (R_d,DB_d,n,List0_d);
cudaMemcpy(List, List_d, n,cudaMemcpyDeviceToHost);
QuickSort(List)
cudaMemcpy(List2_d List, 200, cudaMemcpyHostToDevice);
cudaMemcpy(L_d L, (200*n), cudaMemcpyHostToDevice);
SS2_kernel <<< No_blocks, No_threads >>> (List2_d,200, DB_d,n, L_d,n);
cudaDeviceSynchronize();
Fusion_kernel <<<No_blocks, No_threads >>> (L_d,200,n);
cudaDeviceSynchronize();
cudaMemcpy(L, L_d,(9*n),cudaMemcpyDeviceToHost);
write (L) to text files
end TSS

Figure 3.11: Pseudocode of the Parallel TSS algorithm using CUDA method 2

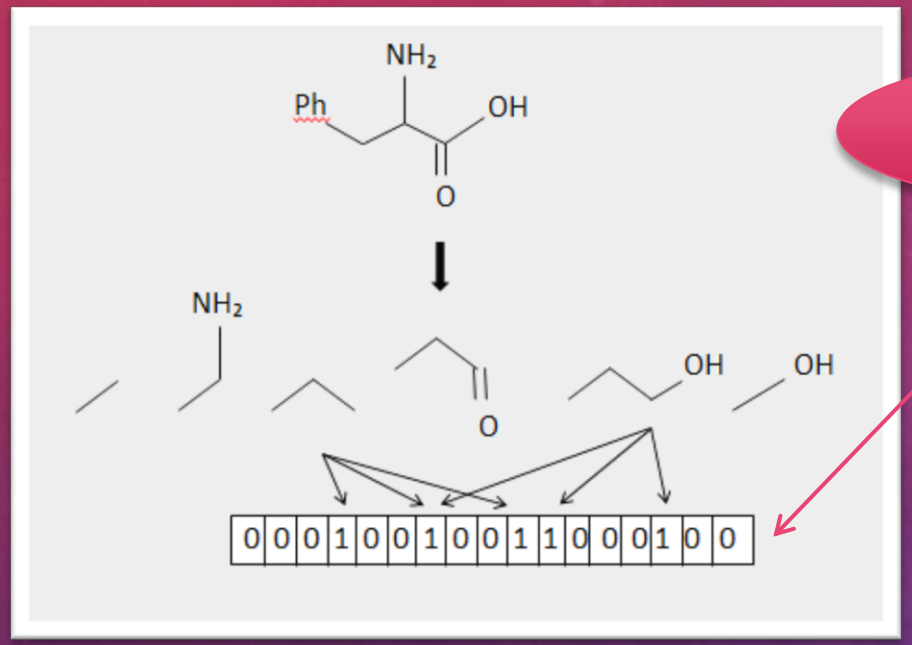
TSS ON CUDA & OPENMP

MARWAH HAITHAM AL-LAILA, NURUL HASHIMAH AHAMED HASSAIN MALIM

Numbers of NN's SS results being fused	Execution time (milliseconds)			
	Sequential	OpenMP	CUDA1	CUDA2
5	6346	4287.6	31.894	30.341
10	11822	7363.8	66.792	63.385
15	17260	10888.8	98.846	93.320
20	22602	14083.4	131.548	124.234
30	32656	19314.8	199.846	189.193
40	42732	24291.4	267.704	253.419
50	52592	29010.4	339.664	322.185
100	99314	45289.8	703.630	671.054
200	192064	75285.4	1460	1392.35

Numbers of NN's SS results being fused	GPU Memory used for CUDA 1 (MB)	GPU Memory used for CUDA 2 (MB)
5	444.20	805.39
10	444.98	807.35
15	445.76	809.30
20	446.54	811.26
30	447.32	815.17
40	448.11	819.08
50	448.89	822.99
100	449.67	842.55
200	450.45	881.67

STRUCTURAL REPRESENTATION



Binary Descriptor

ECFP

Non-Binary Descriptor

ECFC

0 0 0 3 0 0 2 0 0 1 1 0 0 0 4 0 0

Weighted by the frequency value of each fragments

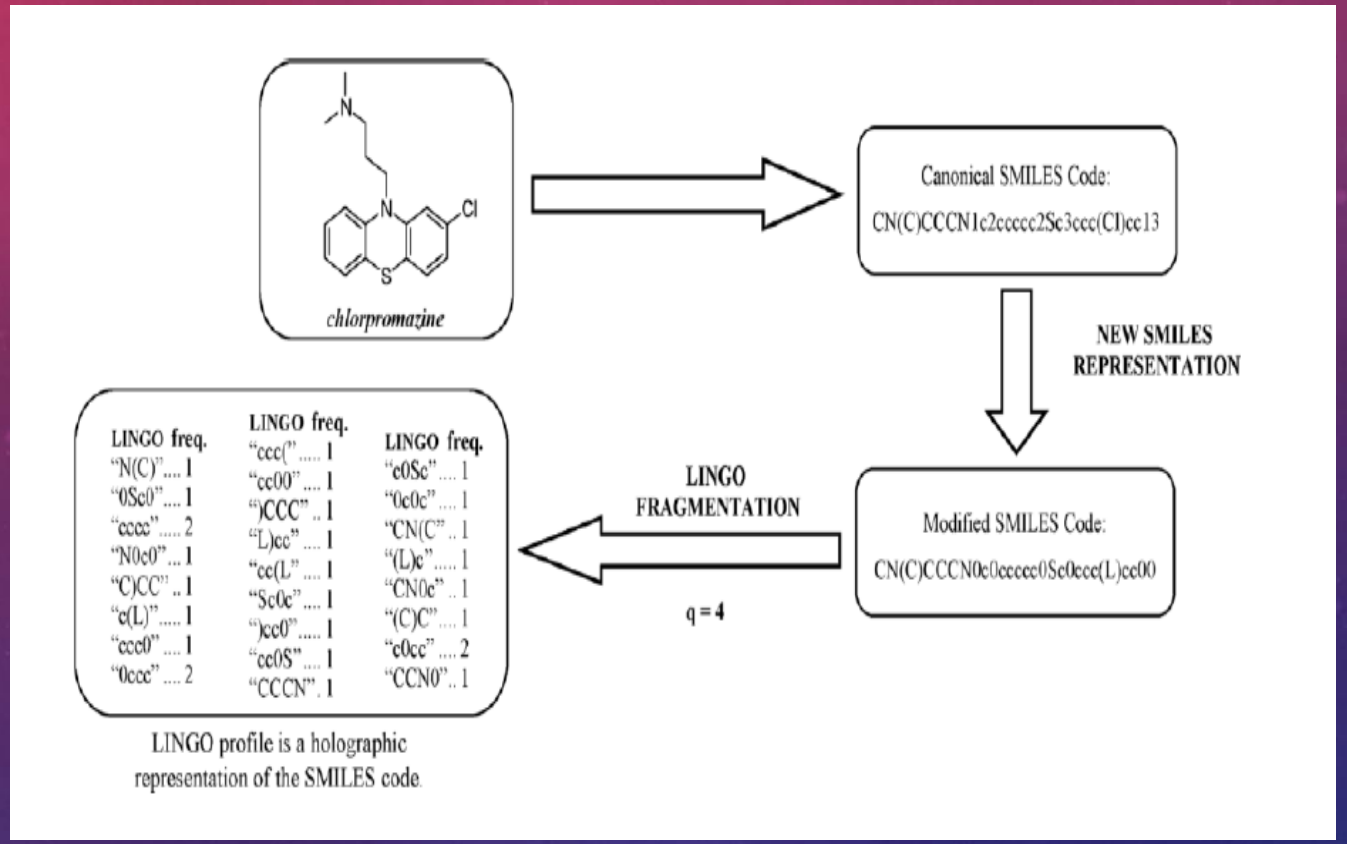
SRECFC

0 0 0 1.73 0 0 1.41 0 0 1 1 0 0 0 2 0 0

Weighted by the square root of the frequency value of each fragments

LATEST WORK .. LINGO

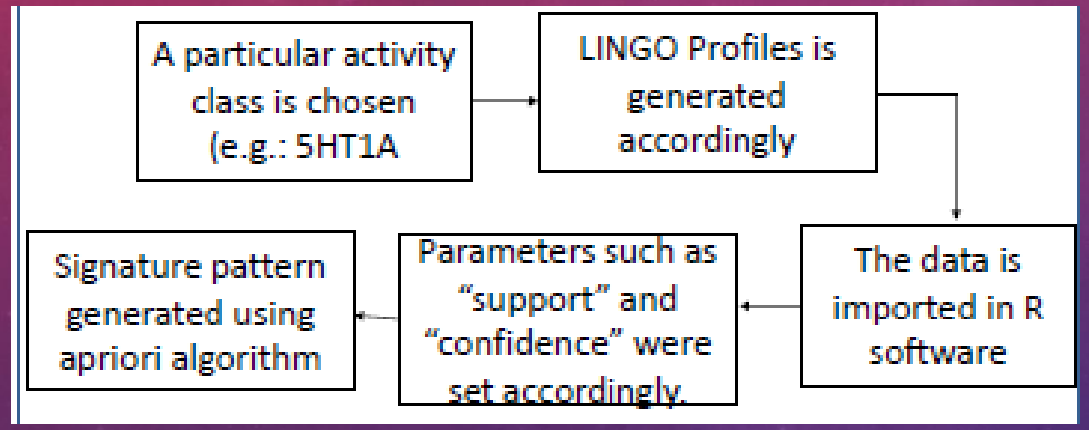
MUHAMMAD JAZIEM JAVEED, NORAZZAWANI AZMAN, FATEN UMIDALILA, SYUHADAH RAMLI, NURUL HASHIMAH AHAMED HASSAIN MALIM



LINGO Generation Process (Vidal et al 2005)

ASSOCIATION RULE MINING

MUHAMMAD JAZIEM JAVED, NURUL HASHIMAH AHAMED HASSAIN MALIM



Rules	Support	Confidence	Lift
{C(=O} => {(=O)}	0.638365	1	1.397802
{(=O)} => {C(=O}	0.638365	0.892308	1.397802
{0ccc} => {c0cc}	0.79717	0.998031	1.090632
{c0cc} => {0ccc}	0.79717	0.871134	1.090632

Figure 5. Interesting rules generated using COX activity class.

Rules	Support	Confidence	Lift
{C(=O} => {(=O)}	0.631347	1	1.36036
{(=O)} => {C(=O}	0.631347	0.858859	1.36036
{0ccc,ccc0} => {cccc}	0.618102	0.949153	1.318914
{0ccc,c0cc,ccc0} => {cccc}	0.618102	0.949153	1.318914
{cccc} => {ccc0}	0.618102	0.858896	1.314459

Figure 6. Interesting rules generated using PKC activity class.

Rules	Support	Confidence	Lift
{c(cc,c0)c,cc(c} => {(cc0}	0.662778	1	1.326301
{c(cc,cc(c,Cc0c} => {(cc0}	0.725345	1	1.326301
{c(cc,c0)c,cc0)} => {(cc0}	0.662778	1	1.326301
{c(cc,cc0),Cc0c} => {(cc0}	0.715801	1	1.326301
{c(cc,c0)c,ccc(} => {(cc0}	0.660657	1	1.326301

Figure 7. Interesting rules generated using AT1 activity class.

Rules	Support	Confidence	Lift
{(Cc0,C(O)} => {(O)C}	0.610667	0.995652	1.427799
{(Cc0,C(O),Cc0c} => {(O)C}	0.610667	0.995652	1.427799
{(Cc0,C(O),cccc} => {(O)C}	0.609333	0.995643	1.427786
{(Cc0,0ccc,C(O)} => {(O)C}	0.609333	0.995643	1.427786
{(Cc0,C(O),c0cc} => {(O)C}	0.609333	0.995643	1.427786

Figure 8. Interesting rules generated using HIVP activity class.

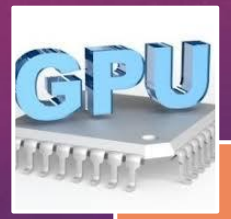
PRESENT & FUTURE

Databases

- MDL Drug Data Report (MDDR)
- ChEMBL
- NADI

Applied Method

- Similarity Searching
- Data Fusion
- Machine Learning
- Deep Learning
- Association Rule Mining



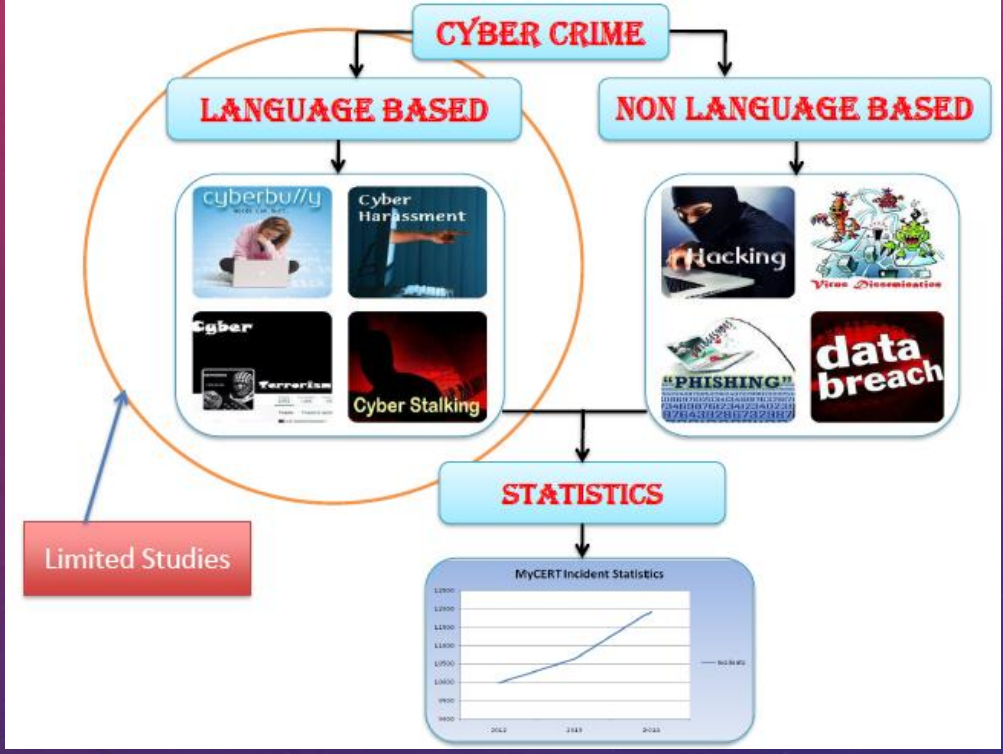
Processing Platform

- Normal Processing
- Parallel Processing
  - Multi-core
    - OpenMP
  - Many-cores
    - GPU - Tesla C2060

CYBERCRIMINAL PERSONALITY DETECTION ON ONLINE SOCIAL NETWORK

SARAVANAN SAGADEVAN, MUHD BAQIR HAKIM, NURUL IZATI RIDZUWAN, NURUL HASHIMAH AHAMED HASSAIN MALIM

CYBERCRIMINAL PERSONALITY DETECTION ON ONLINE SOCIAL NETWORK



Literature Review :
Psychology + Linguistics + Behavior + Criminology studies

↓

Result

↓

* Personality traits influence criminal to conduct crime activities.
* The writings and used of language strongly correlated with personality traits

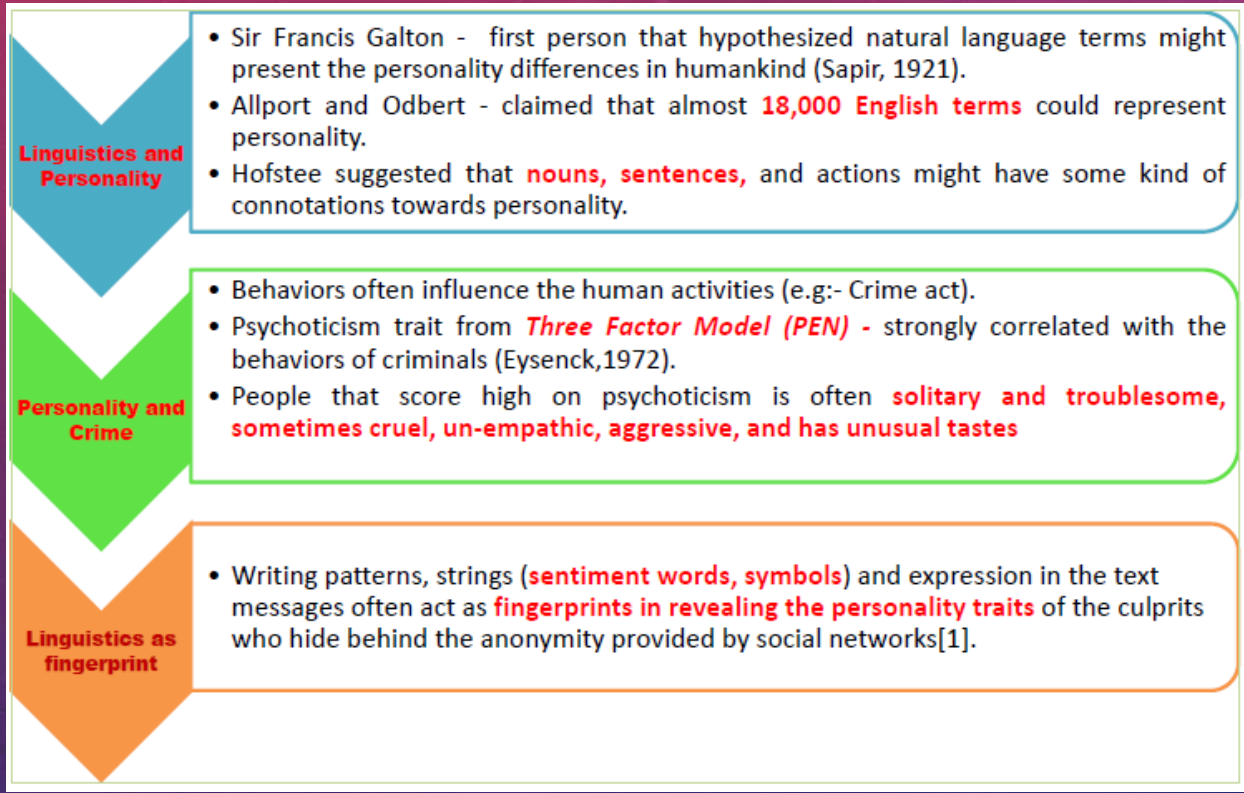
↓

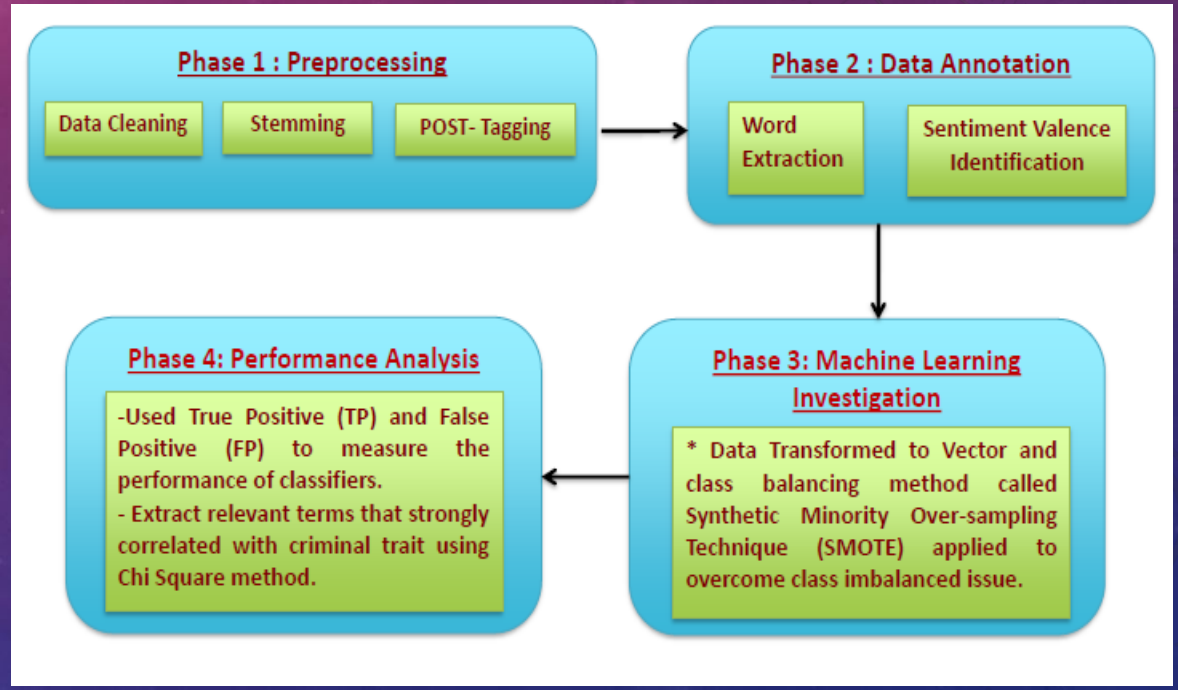
Raised Curiosity

↓

* Whether writing pattern in social networks by cyber criminals could be detect automatically by using personality traits (Psychoticism trait from Three Factor Model).
* If yes, how much better would be the performances of the classifiers be and what are the words or combination of words that may be frequently used by cyber predators.

CYBERCRIMINAL PERSONALITY DETECTION ON ONLINE SOCIAL NETWORK





CYBERCRIMINAL PERSONALITY DETECTION ON ONLINE SOCIAL NETWORK

Performance measurement based on True Positive (TP) and False Positive (FP)						
Cross Validation	3		5		10	
Classifier	TP	FP	TP	FP	TP	FP
ZeroR	53.3	46.7	53.3	46.7	53.3	46.7
NB	80.0	20.0	90.0	10.0	90.0	10.0
KNN	63.3	36.7	56.7	43.3	56.7	43.3
SMO	73.3	26.7	70.0	30.0	86.3	16.7
J48	50.0	50.0	63.3	36.7	70.0	30.0

LATEST ....

- **We are seeking any collaboration with any parties that possess large textual datasets (e.g. from social networks, IRC chat or Dark Web) wrote by the real/actual cyber terrorists and cyber criminals against their victims.**

Thank You!

nurulhashimah@usm.my