

Bit vector description of docked poses¹

D. Rognan, G. Marcou

Bioinformatics of the drug
CNRS UMR7175-LC1
Gilbert Laustriat Institute
F-67400 Illkirch, France

<http://bioinfo-pharma.u-strasbg.fr>

December 4, 2006

¹document version 2.0

Contents

1	Introduction	9
2	Atomic Fingerprint	10
2.1	Definition of an atomic property	10
2.1.1	Hydrophobicity	10
2.1.2	Cationicity	10
2.1.3	Anionicity	11
2.1.4	Aromaticity	11
2.1.5	Hydrogen bond donor	11
2.1.6	Hydrogen bond acceptor	11
2.1.7	Weak hydrogen bond donor	11
2.1.8	Weak hydrogen bond acceptor	12
2.1.9	Metal	12
2.2	Local and non local definitions	12
2.2.1	Aromatic rings	12
2.3	Arithmetic	12
2.4	AFP Application Program Interface	14
2.4.1	SetLcloRng	14
2.4.2	SetHdist	14
2.4.3	SetHangl	15
2.4.4	SetHdngl	15
2.4.5	SetPSdist	15
2.4.6	SetPSangle	15
2.4.7	SetPSdngl	15
2.4.8	SetEFdist	15
2.4.9	SetEFangl	16
2.4.10	SetEFdngl	16
2.4.11	SetHydist	16

2.4.12	SetIondst	16
2.4.13	GetLcloRng	16
2.4.14	GetUnqoMlt	16
2.4.15	GetHdist	17
2.4.16	GetHangl	17
2.4.17	GetHdngl	17
2.4.18	GetPSdist	17
2.4.19	GetPSangle	17
2.4.20	GetPSdngl	17
2.4.21	GetEFdist	18
2.4.22	GetEFangl	18
2.4.23	GetEFdngl	18
2.4.24	GetHydist	18
2.4.25	GetIondst	18
2.4.26	SetIsHD	18
2.4.27	SetIsHA	19
2.4.28	SetIsAr	19
2.4.29	SetIsHy	19
2.4.30	SetIsCat	19
2.4.31	SetIsAnn	19
2.4.32	SetProperties	20
2.4.33	PropReset	20
2.4.34	SetDirections	20
2.4.35	GetIsHD	20
2.4.36	GetIsHA	20
2.4.37	GetIsAr	21
2.4.38	GetIsHy	21
2.4.39	GetIsCat	21
2.4.40	GetIsAnn	21
2.4.41	GetHDdir	21
2.4.42	GetHAdir	21
2.4.43	GetNArodir	22
2.4.44	GetGC	22
2.4.45	Arithmetic operator +	22

3	MFP	23
3.1	Concepts and interface	23
3.2	MFP Application Program Interface	23

3.2.1	Constructor	23
3.2.2	InitResBVPos	24
3.2.3	SetIMolFP	24
3.2.4	SetCav	24
3.2.5	GetCav	24
3.2.6	GetIMolFP	24
3.2.7	ResetResBVPos	25
3.2.8	ResetIMolFP	25
3.2.9	Output stream operator «	25
3.2.10	Output stream operator »	25
4	MAFP	26
4.1	Concepts and interface	26
4.2	MAFP Application Program Interface	26
4.2.1	Constructor	26
4.2.2	InitAtmBVPos	27
4.2.3	SetIMolFP	27
4.2.4	SetLig	27
4.2.5	GetLig	27
4.2.6	GetIMolFP	27
4.2.7	ResetAtmBVPos	28
4.2.8	ResetIMolFP	28
4.2.9	Output stream operator «	28
4.2.10	Output stream operator »	28
5	MyAFP	29
5.1	Presentation and concepts	29
5.2	MyAFP Application Interface	29
5.2.1	constructor	29
5.2.2	SetSmartDist	30
5.2.3	SetSmartA	30
5.2.4	SetIsSmartB	31
5.2.5	GetIsSmartA	31
5.2.6	GetIsSmartB	31
5.2.7	GetSmartCG	31
5.2.8	SetProperties	31
5.2.9	PropReset	31
5.2.10	SetDirections	32

5.2.11	Arithmetic operator +	32
6	MyMFP	33
6.1	Presentation and concepts	33
6.2	MyMFP Application Program Interface	34
6.2.1	constructor	34
6.2.2	SetIMolFP	34
6.2.3	Output stream operator «	34
7	ExtAFP	35
7.1	Presentation and concepts	35
7.2	ExtAFP Application Program Interface	35
7.2.1	constructor	35
7.2.2	SetWHBdist	36
7.2.3	GetWHBdist	36
7.2.4	SetWHBangle	36
7.2.5	GetWHBangle	36
7.2.6	SetWHBdngle	36
7.2.7	GetWHBdngle	36
7.2.8	SetPiCatdist	37
7.2.9	GetPiCatdist	37
7.2.10	SetPiCatangle	37
7.2.11	GetPiCatangle	37
7.2.12	SetPiCatdngle	37
7.2.13	GetPiCatdngle	37
7.2.14	SetPiMetaldist	38
7.2.15	GetPiMetaldist	38
7.2.16	SetIsWHBA	38
7.2.17	SetIsWHBD	38
7.2.18	GetIsWHBA	38
7.2.19	GetIsWHBD	39
7.2.20	SetWGC	39
7.2.21	GetWGC	39
7.2.22	SetWHDdir	39
7.2.23	GetWHDdir	39
7.2.24	SetPiCatGC	39
7.2.25	GetPiCatGC	40
7.2.26	SetPiCatdir	40

7.2.27	GetPiCatdir	40
7.2.28	SetIsMetal	40
7.2.29	GetIsMetal	40
7.2.30	SetMetalGC	40
7.2.31	GetMetalGC	41
7.2.32	SetProperties	41
7.2.33	PropReset	41
7.2.34	SetDirections	41
7.2.35	Arithmetic operator +	41
8	ExtMFP	43
8.1	Presentation and concepts	43
8.2	ExtMFP Application Program Interface	43
8.2.1	constructor	43
8.2.2	SetIMolFP	43
8.2.3	Output stream operator «	44
9	Smallest Ring Library	45
9.1	Presentation and concepts	45
9.2	Smallest Ring application program interface	46
9.2.1	Reset	46
9.2.2	GetSmlstRng	46
9.2.3	GetSeekId	46
9.2.4	GetMaxCnt	46
9.2.5	SetSeekId	46
9.2.6	SetMaxCnt	47
9.2.7	SetSearch	47
9.2.8	FndARng	47
9.2.9	CleanRng	47
10	Medoid tree data organization	48
10.1	Presentation and concepts	48
10.2	Tree medoid application program interface	49
10.2.1	DistNode	49
10.2.2	MatrixDivide	49
10.2.3	treemedoid	49
10.3	Cluster and Nexus formatted trees application program interface	50
10.4	NexusTreeOut	51

10.5	NexusFileIn	51
10.6	NexusTreeIn	52
11	Machine learning: Bayesian classifier	53
11.1	Presentation and concepts	53
11.2	Bayes application program interface	54
11.2.1	Constructor	54
11.2.2	Reset	54
11.2.3	SetAlgo	54
11.2.4	GetAlgo	54
11.2.5	Training	54
11.2.6	SetInferXia	55
11.2.7	SetInferBayes	56
11.2.8	SetOnSupport	56
11.2.9	SetOffSupport	57
11.2.10	SetSupport	57
11.2.11	GetOnSupport	58
11.2.12	GetOffSupport	58
11.2.13	SetOnSupportMin	58
11.2.14	SetOffSupportMin	58
11.2.15	GetOnSupportMin	58
11.2.16	GetOffSupportMin	59
11.2.17	SetOnInfer	59
11.2.18	SetOffInfer	59
11.2.19	GetOnInfer	59
11.2.20	GetOffInfer	59
11.2.21	Predict	60
12	Machine learning: SVM	61
12.1	Presentation and concepts	61
12.2	SVM application program interface	61
12.2.1	Constructor	61
12.2.2	Destructor	62
12.2.3	Reset	62
12.2.4	AddBVTrain	62
12.2.5	SetBVTrain	62
12.2.6	GetBVTrain	62
12.2.7	AddClasses	62

12.2.8	SetClasses	63
12.2.9	GetClasses	63
12.2.10	SetSVMProblem	63
12.2.11	GetSVMProblem	63
12.2.12	SetSVMParameter	63
12.2.13	GetSVMParameter	63
12.2.14	Training	64
12.2.15	SetModel	64
12.2.16	GetModel	64
12.2.17	Predict	64
13	Installation	65
13.1	Boost library	65
13.1.1	Boost license	65
13.1.2	Where are Boost libraries being used?	66
13.2	OpenEye	66
13.2.1	Where are OpenEye libraries being used?	67
13.2.2	OpenEye and OpenBabel	67
13.3	C Clustering Library	67
13.3.1	License	68
13.4	libSVM	74
13.4.1	License	74
13.5	Application installation	75
14	Application manual	77
14.1	IFP, IAFP, MyIFP, ExtIFP	77
14.1.1	Command line	78
14.1.2	Command file	80
14.2	LBMFP,ExtLBMFP	81
14.2.1	Command Line	81
14.2.2	Command file	83
14.3	Bayes	83
14.3.1	Command Line	84
14.3.2	Command file	86
14.4	SVM	86
14.4.1	Command Line	86
14.4.2	Command file	89
14.5	T2C	90

15 Interpretations of the output fingerprints	91
15.1 Flag and geometric descriptor assignment	91
15.2 Geometry	91
15.3 Fingerprint structure	93
16 Conclusion	94
17 Support	95

Chapter 1

Introduction

Docked poses are typically scored and ranked according to energy-based scoring functions then filtered on geometric criteria, visual analysis and a bit of black magic. Upon filtering, one usually relies on the similarity of a docked pose to one that is known or believed to be of good quality. This is usually done by visual inspection of 3D or 2D representations of the docked poses. Another option is to check if similar molecular features are in the same environment. This is the goal of the present library and the present applications.

Chapter 2

Atomic Fingerprint

This library describes the molecular features associated to each atom of a ligand. Thus, we must define an Atomic Fingerprint for each atom of interest. The library defines also some rules to evaluate how two atoms interact according to these features. This defines a C++ class called AFP.

2.1 Definition of an atomic property

An Atomic Fingerprint is defined for one atom only. A property is a set of variables that defines if a given interaction involving this atom is possible or not and its possible characteristics. For example an atom that can be involved in an hydrogen bond will have a flag saying if it is an hydrogen donor or acceptor, a point from which to evaluate a distance to its counterpart, a vector to possibly define the angle of the interaction.

Properties are defined in the header of the class. A list of the defined properties follows.

2.1.1 Hydrophobicity

An atom is flagged *hydrophobe* if it is a carbon atom, a sulfur or an halogen (F, Cl, Br, I). The property center is the atom center.

2.1.2 Cationicity

An atom is flagged *cationic* if it carries a positive formal charge. The property center is the center of the atom.

2.1.3 Anionicity

An atom is flagged *anionic* if it carries a negative formal charge. The property center is the atom center.

2.1.4 Aromaticity

A flagged *aromatic* atom is defined as an atom that is not non-aromatic. The definition of non-aromatic atoms is the one given by OpenEye in the OEChem library¹.

The property center can be either the center of the aromatic ring the atom belong to or the center of the atom. The property direction is the vector normal to the aromatic plane defined by the atom and two of its aromatic neighbors.

2.1.5 Hydrogen bond donor

An atom is flagged an *hydrogen bond donor* if it is an hydrogen covalently linked to either an oxygen, a nitrogen or a sulfur atom. The property center is the heavy atom center the hydrogen is linked to. The property direction is the heavy atom to hydrogen vector.

2.1.6 Hydrogen bond acceptor

An atom is flagged an *hydrogen bond acceptor* if has a negative formal charge or is either an oxygen or a nitrogen with a null formal charge. The property center is the atom center. It has no direction.

2.1.7 Weak hydrogen bond donor

A *weak hydrogen bond donor* flagged atom is an hydrogen covalently bond to either an sp, sp² or aromatic carbon. The interaction center is the heavy atom center. The interaction direction is the heavy atom to hydrogen direction.

¹www.eyesopen.com

2.1.8 Weak hydrogen bond acceptor

A *weak hydrogen bond acceptor* flagged atom is either a normal acceptor or a pi-electronic cloud (carbon sp, sp² or aromatic). The interaction center is the atom center. It has no direction.

2.1.9 Metal

A *metal* flagged atom is either a Mg, Ca, Mn, Fe, Co, Ni, Zn, Cu or Cd. The interaction center is the atom center.

2.2 Local and non local definitions

Some properties ensue from the chemical group the considered atom belong to. In that case the same property exists for each atom of this chemical group. Therefore, the center of the interaction can lay away from the center of the atom.

The concept of non-local is defined in opposition to the one of local. A local definition has a center on the current atom and at most one direction associated.

2.2.1 Aromatic rings

This is the first case of such kind of definitions. Aromatic interactions are directional. Two aromatic rings interact either as “edge to face” or “face to face” depending on the angle made by their planes. The definition of the center of the interaction is not unique.

If the definition is local, the center of the interaction is the current aromatic atom. If the definition is non-local, the smallest aromatic ring the atom belong to is identified. The center of the interaction is the geometric center of the ring.

2.3 Arithmetic

Two Atomic Fingerprints can be added. In that case possible interactions for both atoms are tested according to the data stored in the Atomic Fingerprints.

If an atom is found to be an *hydrogen bond acceptor* and the other an *hydrogen bond donor*, the distance between them is tested against the maximum distance allowed for hydrogen bonding. The angle between the directions of both Atomic Fingerprints will be tested against the expected angle of an hydrogen bond, plus or minus the tolerance.

If both atoms are found to be *aromatic*, the distance between them is tested against the maximum distance allowed for “face to face” and “edge to face” interactions. The angle between the directions of both Atomic Fingerprints will be tested against the expected angle of these interactions, plus or minus the tolerance.

If both atoms are found to be *hydrophobic*, the distance between them is tested against the maximum distance allowed for hydrophobic interactions.

If both atoms are found to be *ionic* and of opposite charge, the distance between them is tested against the maximum distance allowed for “ionic” interactions.

The result of the operation is an integer that is a product of prime numbers, each prime number corresponding to a particular interaction.

- 1: no interaction
- 2: Hydrophobic
- 3: Face to face (Pi stacking)
- 5: Edge to face
- 7: H-bond Donor-Acceptor
- 11: H-bond Acceptor-Donor
- 13: Ionic interaction + -
- 17: Ionic interaction - +

For example $91 = 7 * 13$ corresponds to an interaction that is an donor-acceptor hydrogen bond aided by an ionic interaction. Note that the operator is not commutative.

There is an extension to those referenced interactions:

- 19: Weak H-bond Donor-Acceptor

23: Weak H-bond Acceptor-Donor

29: Pi-Cation

31: Metal coordination

Why the hell is it so complicated? Using prime numbers to code interactions is complicated and asks for prime number decomposition that is a very difficult task. But it has some advantages. First, combining several atomic interactions is simply an integer multiplication. Redundancy of the interactions is not lost. Second, the result of the operation is device independent and can be stored and manipulated without care. Third, the output of the operation is a standard type integer, so it is easy to manage. Fourth, if someone adds a new interaction type, it will simply append to the library (by modifying the source or by inheritance). The corresponding properties and descriptors will define a new prime number. This modification will not affect programs that already use the library (unless wanted so by the programmer).

2.4 AFP Application Program Interface

This is a description of the class AFP.

2.4.1 SetLcloRng

Using local or non-local aromatic definition?

```
static inline void SetLcloRng(bool b)
```

2.4.2 SetHdist

Set the hydrogen bond maximum length.

```
static inline void SetHdist(double d)
```

2.4.3 SetHangl

Set the hydrogen bond angle.

```
static inline void SetHangl(double a)
```

2.4.4 SetHdngl

Set the hydrogen bond angle tolerance.

```
static inline void SetHdngl(double a)
```

2.4.5 SetPSdist

Set the Π -stacking maximum distance.

```
static inline void SetPSdist(double d)
```

2.4.6 SetPSangle

Set the Π -stacking angle.

```
static inline void SetPSangl(double a)
```

2.4.7 SetPSdngl

Set the Π -stacking angle tolerance.

```
static inline void SetPSdngl(double a)
```

2.4.8 SetEFdist

Set the edge-to-face maximum distance.

```
static inline void SetEFdist(double d)
```


2.4.9 SetEFangl

Set the edge to face angle.

```
static inline void SetEFangl(double a)
```

2.4.10 SetEFdngl

Set the edge to face angle tolerance.

```
static inline void SetEFdngl(double a)
```

2.4.11 SetHydist

Set the Hydrophobic interaction distance.

```
static inline void SetHydist(double d)
```

2.4.12 SetIondst

Set the ionic interaction distance.

```
static inline void SetIondst(double d)
```

2.4.13 GetLcloRng

Tell if using local or non-local aromatic definition?

```
static inline bool GetLcloRng()
```

2.4.14 GetUnqoMlt

Tell if using local or non-local hydrogen bond donor definition?

```
static inline bool GetUnqoMlt()
```

2.4.15 GetHdist

Get the hydrogen bond maximum length.

```
static inline double GetHdist()
```

2.4.16 GetHangl

Get the hydrogen bond angle.

```
static inline double GetHangl()
```

2.4.17 GetHdngl

Get the hydrogen bond angle tolerance.

```
static inline double GetHdngl()
```

2.4.18 GetPSdist

Get the Π -stacking maximum distance.

```
static inline double GetPSdist()
```

2.4.19 GetPSangle

Get the Π -stacking angle.

```
static inline double GetPSangl()
```

2.4.20 GetPSdngl

Get the Π -stacking angle tolerance.

```
static inline double GetPSdngl()
```

2.4.21 GetEFdist

Get the edge-to-face maximum distance.

```
static inline double GetEFdist()
```

2.4.22 GetEFangl

Get the edge to face angle.

```
static inline double GetEFangl()
```

2.4.23 GetEFdngl

Get the edge to face angle tolerance.

```
static inline double GetEFdngl()
```

2.4.24 GetHydist

Get the Hydrophobic interaction distance.

```
static inline double GetHydist()
```

2.4.25 GetIondst

Get the ionic interaction distance.

```
static inline double GetIondst()
```

2.4.26 SetIsHD

Set or find if an atom is an hydrogen bond donor.

```
inline void SetIsHD(bool b)
void SetIsHD(OEAtomBase *X)
```

2.4.27 SetIsHA

Set or find if an atom is an hydrogen bond acceptor.

```
inline void SetIsHA(bool b)
void SetIsHA(OEAtomBase *X)
```

2.4.28 SetIsAr

Set or find if an atom is aromatic.

```
inline void SetIsAr(bool b)
inline void SetIsAr(OEAtomBase *X)
```

2.4.29 SetIsHy

Set or find if an atom is hydrophobe.

```
inline void SetIsHy(bool b)
void SetIsHy(OEAtomBase *X)
```

2.4.30 SetIsCat

Set or find if an atom is cationic.

```
inline void SetIsCat(bool b)
inline void SetIsCat(OEAtomBase *X)
```

2.4.31 SetIsAnn

Set or find if an atom is anionic.

```
inline void SetIsAnn(bool b)
inline void SetIsAnn(OEAtomBase *X)
```

2.4.32 SetProperties

Automatically detect all properties. It does not set the center or direction of the properties.

```
void SetProperties(OEAtomBase *X)
```

2.4.33 PropReset

Cancel or reset all properties.

```
void PropReset();//Reset properties
```

2.4.34 SetDirections

Automatically find center and directions of the properties. The properties must have been detected before.

```
void SetDirections(OEMolBase &mol, OEAtomBase *X);
```

2.4.35 GetIsHD

Get if an atom is an hydrogen bond donor.

```
inline bool GetIsHD()
```

2.4.36 GetIsHA

Get if an atom is an hydrogen bond acceptor.

```
inline bool GetIsHA()
```

2.4.37 GetIsAr

Get if an atom is aromatic.

```
inline bool GetIsAr()
```

2.4.38 GetIsHy

Get if an atom is hydrophobe.

```
inline bool GetIsHy(bool b)
```

2.4.39 GetIsCat

Get if an atom is cationic.

```
inline bool GetIsCat()
```

2.4.40 GetIsAnn

Get if an atom is anionic.

```
inline bool GetIsAnn()
```

2.4.41 GetHDdir

Get hydrogen bonding donor direction.

```
inline void GetHDdir(double X[3])
```

2.4.42 GetHAdir

Get hydrogen bonding acceptor directions.

```
inline void GetHAdir(vector<double> X[3])
```

2.4.43 GetNArodir

Get the normal to the aromatic ring.

```
inline void GetNArodir(double X[3])
```

2.4.44 GetGC

Get the property center. Note that if a carbon is aromatic this point may not be the center of the atom.

```
inline void GetGC(double X[3])
```

2.4.45 Arithmetic operator +

This operator defines how two Atomic Fingerprints interact with each other.

The result of the operation is an integer that is a product of prime numbers, each prime number corresponding to a particular interaction.

- 1: no interaction
- 2: Hydrophobic
- 3: Face to face (Pi stacking)
- 5: Edge to face
- 7: H-bond Donor Acceptor
- 11: H-bond Acceptor Donor
- 13: Ionic interaction + -
- 17: Ionic interaction - +

For example $91 = 7 * 13$ corresponds to an interaction that is a donor-acceptor hydrogen bond aided by an ionic interaction. Note that the operator is not commutative.

```
friend int operator + (const AFP&, const AFP&)
```

Chapter 3

MFP

This library is based on the Atomic Fingerprint concept. Its goal is to derive the possible interactions from Atomic Fingerprints generated by two groups of molecules. One group will be called the cavity, the other one the ligand.

3.1 Concepts and interface

The MFP is attached to a molecule called the cavity. It stores a vector of Atomic Fingerprints associated with each atom of this molecule. It is also able, given a second molecule called the ligand, to store in a vector of bit vector the list of recorded interactions between the cavity and the ligand. The cavity is thought to be a protein, so the base unit is the residue and the vector of bit vector is mapped to residue indexes.

3.2 MFP Application Program Interface

This is a description of the class MFP. Each MFP object correspond to one cavity.

3.2.1 Constructor

This particular form of the constructor allows to build the object with its related cavity.

```
MFP::MFP(OEMol &mol);
```


3.2.2 InitResBVPos

The cavity is thought to be a protein, so the base unit is the residue and the vector of bit vector is mapped to residue index. This method allows to initialize the map.

```
void InitResBVPos();
```

3.2.3 SetIMolFP

Set the Molecular Interaction Fingerprint based on the cavity of the MFP and a ligand.

```
inline void SetIMolFP(OEBitVector& OEBV){  
void SetIMolFP(OEMol &lig);
```

3.2.4 SetCav

Set the cavity. This task can be automatically done by calling the appropriate constructor.

```
inline void SetCav(OEMol &mol)
```

3.2.5 GetCav

Get the cavity.

```
inline OEMol GetCav()
```

3.2.6 GetIMolFP

Get the molecular finger print vector of bit vector.

```
inline OEBitVector GetIMolFP()
```

3.2.7 ResetResBVPos

Reset the map of the residue indexes to the vector of bit vector.

```
inline void ResetResBVPos()
```

3.2.8 ResetIMolFP

Reset the vector of bit vector.

```
inline void ResetIMolFP()
```

3.2.9 Output stream operator «

This operator is overloaded to allow output of the object. It will create an output stream containing 0 or 1 for each bit of the vector of bit vector. Each subset of 7 bits correspond to a residue. The residues appear in the list in the same order they were used to build the object.

If the user set the length of an interaction to 0 the corresponding bits will disappear. In that case residue bit vectors will be less than 7 bits long.

Above the bit vectors, a line will give the residue name of the corresponding bit vector subset. Each residue name is separated by a column (|). If the bit vector becomes too short, the alignment between the bit vector subset and the corresponding residue name might fail.

```
inline ostream& operator << (ostream &strm, MFP &MolFP)
```

3.2.10 Output stream operator »

This operator is overloaded to allow input of the bit vector of the object. It converts a character stream to bit vector finger print of the object, 0 for false, 1 for true.

```
inline istream& operator >> (istream &strm, MFP &MolFP)
```

Chapter 4

MAFP

This class is based on the Atomic Fingerprint concept. Its goal is to derive the possible interactions from Atomic Fingerprints from two groups of molecules. One group will be called the ligand, the other one the cavity. By opposition to MFP described above, AFP interactions are followed atom by atom.

4.1 Concepts and interface

The MAFP is attached to the molecule called the ligand. It stores a vector of Atomic Fingerprints associated with each atom of this molecule. It is also able, given a second molecule called the cavity, to store in a vector of bit vector the list of recorded interactions between the cavity and the ligand. Here the focus is on the ligand atoms, so the base unit of the interaction finger print bit vector is the ligand atom to which it is mapped.

4.2 MAFP Application Program Interface

This is a description of the class MAFP.

4.2.1 Constructor

The constructor allows to build the instance of the class with the ligand.

```
MAFP::MAFP();  
MAFP::MAFP(OEMol &mol);
```

4.2.2 InitAtmBVPos

The base unit is the ligand atom and the vector of bit vector is mapped to atom index. This method allows to initialize the map.

```
void InitAtmBVPos();
```

4.2.3 SetIMolFP

Set the Molecular Interaction Fingerprint based on the ligand of the MAFP and a cavity.

```
inline void SetIMolFP(OEBitVector& OEBV){  
void SetIMolFP(OEMol &lig);
```

4.2.4 SetLig

Set the ligand. This task can be automatically done by calling the appropriate constructor.

```
inline void SetLig(OEMol &mol)
```

4.2.5 GetLig

Get the ligand.

```
inline OEMol GetLig()
```

4.2.6 GetIMolFP

Get the molecular finger print vector of bit vector.

```
inline OEBitVector GetIMolFP()
```

4.2.7 ResetAtmBVPos

Reset the map of the atom indexes to the vector of bit vector.

```
inline void ResetAtmBVPos()
```

4.2.8 ResetIMolFP

Reset the vector of bit vector.

```
inline void ResetIMolFP()
```

4.2.9 Output stream operator «

This operator is overloaded to allow output of the object. It will create an output stream containing 0 or 1 for each bit of the vector of bit vector. Each subset of 7 bits correspond to a ligand atom. The atoms appear in the list in the same order they were used to build the object.

If the user set the length of an interaction to 0 the corresponding bits will disappear. In that case residue bit vectors will be less than 7 bits long.

Above the bit vectors, a line will give the atom name of the corresponding bit vector subset. Each atom name is separated by a column (|). If the bit vector becomes too short, the alignment between the bit vector subset and the corresponding atom name might fail.

```
inline ostream& operator << (ostream &strm, MAFP &MolFP)
```

4.2.10 Output stream operator »

This operator is overloaded to allow input of the bit vector of the object. It converts a character stream to bit vector finger print of the object, 0 for false, 1 for true.

```
inline istream& operator >> (istream &strm, MAFP &MolFP)
```

Chapter 5

MyAFP

This class is conceived as an example of inheritance of AFP class for user-specific fingerprint design. In this example, this fingerprint is designed for limited SMARTS support. That is adding a bit to detect whether to groups defined by SMARTS are closer than a given threshold.

5.1 Presentation and concepts

Class inheritance is a powerful concept in object oriented languages like C++, Java, Ruby or Python. It allows to factorize the common variables and methods of a set of objects. It allow also to build upon, or extend old working data structures without even looking in the source code.

This SMARTS support on another hand allows for a qualitative control of the fingerprints produced by the program. It can be seen also as an experiment toward a full SMARTS definition of the fingerprints. Doing so is a lot more complicated since it implies the creation of a grammar of interaction definition. This is a research project of its own.

5.2 MyAFP Application Interface

This is a description of the class MyAFP, inheriting AFP.

5.2.1 constructor

Use the default constructor.

```
MyAFP(){}  

```

5.2.2 SetSmartDist

Set the inter-SMARTS group maximum distance.

```
static inline void SetSmartDist(double d)
```

5.2.3 SetSmartA

Set the first query molecule (OEqMol) based on a SMARTS string.

```
static inline bool SetSmartA(string&)
```

```
\end{verbaim}
```

If the SMARTS is ill conditioned the method return false.

```
\subsection{SetSmartB}
```

Set the second query molecule (OEqMol) based on a SMARTS string.

```
\begin{verbatim}
```

```
static inline bool SetSmartB(string&){
```

```
\end{verbatim}
```

If the SMARTS is ill conditioned the method return false.

```
\subsection{SetIsSmartA}
```

Set if the atom is a part of the first SMARTS.

```
\begin{verbatim}
```

```
inline void SetIsSmartA(bool b)
```

```
void SetIsSmartA(OEMolBase&, OEAtomBase*)
```

5.2.4 SetIsSmartB

Set if the atom is a part of the second SMARTS.

```
inline void SetIsSmartB(bool b)
void SetIsSmartB(OEMolBase&, OEAtomBase* X)
```

5.2.5 GetIsSmartA

Get if the atom is a part of the first SMARTS.

```
inline bool GetIsSmartA()
```

5.2.6 GetIsSmartB

Get if the atom is a part of the first SMARTS.

```
inline bool GetIsSmartB()
```

5.2.7 GetSmartCG

Get the interaction center coordinates.

```
inline void GetSmartCG(double CG[3])
```

5.2.8 SetProperties

Set all the potential interaction flags.

```
void SetProperties(OEMolBase&, OEAtomBase*)
```

5.2.9 PropReset

Reset all potential interactions flags.

```
void PropReset()
```


5.2.10 SetDirections

Set all potential interactions centers and directions.

```
void SetDirections(OEMolBase&, OEAtomBase*);
```

5.2.11 Arithmetic operator +

This operator defines how two Atomic Fingerprints interact with each other.

The result of the operation is an integer that is a product of prime numbers, each prime number corresponding to a particular interaction.

- 1: no interaction
- 2: Hydrophobic
- 3: Pi stacking
- 5: Edge to face
- 7: H-bond Donor Acceptor
- 11: H-bond Acceptor Donor
- 13: Ionic interaction + -
- 17: Ionic interaction - +
- 19: SMARTS interactions

For example $91 = 7 * 13$ corresponds to an interaction that is a donor-acceptor hydrogen bond aided by an ionic interaction. Note that the operator is not commutative.

```
friend int operator + (const MyAFP&, const MyAFP&);
```

Chapter 6

MyMFP

This class is conceived as an example of inheritance of MFP class for user-specific fingerprint design. In this example, this fingerprint is designed for limited SMARTS support. That is adding a bit to detect whether to groups defined by SMARTS are closer than a given threshold.

6.1 Presentation and concepts

Class inheritance is a powerful concept in object oriented languages like C++, Java, Ruby or Python. It allows to factorize the common variables and methods of a set of objects. It allow also to build upon, or extend old working data structures without even looking in the source code.

This SMARTS support on another hand allows for a qualitative control of the fingerprints produced by the program. It can be seen also as an experiment toward a full SMARTS definition of the fingerprints. Doing so is a lot more complicated since it implies the creation of a grammar of interaction definition. This is a research project of its own.

As MFP, MyMFP is attached to a molecule called the cavity thought to be a protein. It stores a vector of Atomic Fingerprints associated with each atom of this molecule. Given a second molecule called the ligand, it stores in a vector of bit vectors the list of recorded interactions between the cavity and the ligand. As the cavity is thought to be a protein, the base unit is the residue and the vector of bit vectors is mapped to the residues indexes.

6.2 MyMFP Application Program Interface

This is a description of the class MyMFP, inheriting MFP.

6.2.1 constructor

The constructor allows creation of the object while directly setting the cavity.

```
MyMFP(){}  
MyMFP(OEMol &mol):MFP(mol){}
```

6.2.2 SetIMolFP

Set the Molecular Interaction Fingerprint based on the ligand of the AFP and a cavity.

```
void SetIMolFP(OEMol &lig)
```

6.2.3 Output stream operator «

This operator is overloaded to allow output of the object. It will create an output stream containing 0 or 1 for each bit of the vector of bit vector. Each subset of 8 bits correspond to a cavity residue. The residues appear in the list in the same order they were used to build the object.

If the user set the length of an interaction to 0, the corresponding bits will disappear. In that case residue bit vectors will be less than 8 bits long.

Above the bit vectors, a line will give the residue name of the corresponding bit vector subset. Each residue name is separated by a column (|). If the bit vector becomes too short, the alignment between the bit vector subset and the corresponding residue name might fail.

```
inline ostream& operator << (ostream &strm, MyMFP &MolFP)
```

Chapter 7

ExtAFP

The ExtAFP class inherits the AFP class. It is designed to extend the AFP class to account for more interactions.

7.1 Presentation and concepts

This class is an extension of the older AFP class. It allows accounting for weak hydrogen bonding, pi-cation and metal interactions.

Weak hydrogen bonding are intended to be hydrogen bonds involving carbon bond hydrogen as donor and/or pi electronic systems as acceptors.

Pi-cation interactions are, for most of it, explained by quadrupole-monopole electrostatic interactions between aromatic systems and cations.

Metal interactions are to be understood as in metal coordination chemistry. This phenomenon explains the “double salt” behavior of metals.

7.2 ExtAFP Application Program Interface

This is a description of the class ExtAFP, inheriting AFP.

7.2.1 constructor

The constructor allows creation of the object. It resets all private variable.

```
ExtAFP()
```

7.2.2 SetWHBdist

Set Weak Hydrogen bond distance.

```
static inline void SetWHBdist(double d)
```

7.2.3 GetWHBdist

Get Weak Hydrogen bond distance.

```
static inline double GetWHBdist()
```

7.2.4 SetWHBangle

Set Weak Hydrogen bond angle.

```
static inline void SetWHBangle(double d)
```

7.2.5 GetWHBangle

Get Weak Hydrogen bond angle.

```
static inline double GetWHBangle()
```

7.2.6 SetWHBdngle

Set Weak Hydrogen bond deviation angle.

```
static inline void SetWHBdngle(double d)
```

7.2.7 GetWHBdngle

Get Weak Hydrogen bond deviation angle.

```
static inline double GetWHBdngle()
```

7.2.8 SetPiCatdist

Set Pi-Cation distance.

```
static inline void SetPiCatdist(double d)
```

7.2.9 GetPiCatdist

Get Pi-Cation distance.

```
static inline double GetPiCatdist()
```

7.2.10 SetPiCatangle

Set Pi-Cation angle.

```
static inline void SetPiCatangle(double d)
```

7.2.11 GetPiCatangle

Get Pi-Cation angle.

```
static inline double GetPiCatangle()
```

7.2.12 SetPiCatdngle

Set Pi-Cation deviation angle.

```
static inline void SetPiCatdngle(double d)
```

7.2.13 GetPiCatdngle

Get Pi-Cation deviation angle.

```
static inline double GetPiCatdngle()
```

7.2.14 SetPiMetaldist

Set coordination metal distance.

```
static inline void SetMetaldist(double d)
```

7.2.15 GetPiMetaldist

Get coordination metal distance.

```
static inline double GetMetaldist()
```

7.2.16 SetIsWHBA

Set if atom is Weak Hydrogen bond acceptor. A weak H-bond acceptor is either a neutral sulfur atom or pi-electronic cloud (sp, sp2 or aromatic).

```
inline void SetIsWHBA(bool b)
void SetIsWHBA(OEAtomBase*)
```

7.2.17 SetIsWHBD

Set if atom is Weak Hydrogen bond donor. A weak H-bond donor is an hydrogen covalently bond to either an sp, sp2 or aromatic carbon. The property center is set to the center of the heavy atom the hydrogen is attached to.

```
inline void SetIsWHBD(bool b)
void SetIsWHBD(OEAtomBase*)
```

7.2.18 GetIsWHBA

Get if atom is Weak Hydrogen bond acceptor.

```
inline bool GetIsWHBA() const
```

7.2.19 GetIsWHBD

Get if atom is Weak Hydrogen bond donor.

```
inline bool GetIsWHBD() const
```

7.2.20 SetWGC

Set the geometric center of weak hydrogen bond interactions.

```
inline void SetWGC(double U[3])
```

7.2.21 GetWGC

Get the geometric center of weak hydrogen bond interactions.

```
inline void GetWGC(double U[3]) const
```

7.2.22 SetWHDdir

Set the directionality of weak hydrogen bond interactions.

```
inline void SetWHDdir(double U[3])
```

7.2.23 GetWHDdir

Get the directionality of weak hydrogen bond interactions.

```
inline void GetWHDdir(double U[3]) const
```

7.2.24 SetPiCatGC

Set the geometric center of Pi-Cation interactions.

```
inline void SetPiCatGC(double U[3])
```


7.2.25 GetPiCatGC

Get the geometric center of Pi-Cation interactions.

```
inline void GetPiCatGC(double U[3]) const
```

7.2.26 SetPiCatdir

Set the directionality of Pi-Cation interactions.

```
inline void SetPiCatdir(double U[3])
```

7.2.27 GetPiCatdir

Get the directionality of Pi-Cation interactions.

```
inline void GetPiCatdir(double U[3]) const
```

7.2.28 SetIsMetal

Set if atom is a metal. A metal is either a Mg, Ca, Mn, Fe, Co, Ni, Zn, Cu or Cd.

```
inline void SetIsMetal(bool b)
void SetIsMetal(OEAtomBase*)
```

7.2.29 GetIsMetal

Get if atom is a metal.

```
inline bool GetIsMetal() const
```

7.2.30 SetMetalGC

Set the geometric center of metal interactions.

```
inline void SetMetalGC(double GC[3])
```

7.2.31 GetMetalGC

Get the geometric center of metal interactions.

```
inline void GetMetalGC(double GC[3]) const
```

7.2.32 SetPropertyes

Set all properties.

```
void SetPropertyes(OEAtomBase*)
```

7.2.33 PropReset

Reset properties.

```
void PropReset()
```

7.2.34 SetDirections

Set Geometries.

```
void SetDirections(OEMolBase&, OEAtomBase*)
```

7.2.35 Arithmetic operator +

Find interactions.

The result is an integer that is the product of prime numbers corresponding to:

- 1: no interaction
- 2: Hydrophobic
- 3: Pi stacking
- 5: Edge to face

- 7: H-bond Donor Acceptor
- 11: H-bond Acceptor Donor
- 13: Ionic interaction + -
- 17: Ionic interaction - +
- 19: Weak H-bond Donor Acceptor
- 23: Weak H-bond Acceptor Donor
- 29: Pi-Cation
- 31: Metal coordination

For example $91 = 7 * 13$ correspond to an interaction that is a donor-acceptor hydrogen bond aided by an ionic interaction. Note that the operator is not commutative.

```
friend int operator + (const ExtAFP&, const ExtAFP&)
```

Chapter 8

ExtMFP

This class extends the older class MFP.

8.1 Presentation and concepts

This class inherits the class MFP. This class is designed to make use of the extension of atomic fingerprints definition through the use of the ExtAFP class.

8.2 ExtMFP Application Program Interface

This is a description of the class ExtMFP, inheriting MFP.

8.2.1 constructor

The constructor allows creation of the object while directly setting the cavity.

```
ExtMFP(){};  
ExtMFP(OEMol &mol):MFP(mol){};
```

8.2.2 SetIMolFP

Set the Molecular Interaction Fingerprint based on a cavity OEMol and ligand OEMol.

```
void SetIMolFP(OEMol &lig)
```

8.2.3 Output stream operator «

This operator is overloaded to allow output of the object. It will create an output stream containing 0 or 1 for each bit of the vector of bit vector. Each subset of 11 bits correspond to a cavity residue. The residues appear in the list in the same order they were used to build the object.

If the user set the length of an interaction to 0, the corresponding bits will disappear. In that case residue bit vectors will be less than 11 bits long.

Above the bit vectors, a line will give the residue name of the corresponding bit vector subset. Each residue name is separated by a column (|). If the bit vector becomes too short, the alignment between the bit vector subset and the corresponding residue name might fail.

```
inline ostream& operator << (ostream &strm, ExtMFP &MolFP){
```

Chapter 9

Smallest Ring Library

OpenEye does not give any method to get the smallest ring a given atom belongs to, because the notion of smallest ring is ambiguous. It is up to the user to find a definition of what precisely is a smallest ring that fits his own application.

9.1 Presentation and concepts

In this application, we want to assign to an aromatic atom, a precise aromatic ring. If it belongs to two fused aromatic rings, some atoms belong equally to several rings. They will participate to the definition of several rings but for our problem, assigning a precise ring to them is irrelevant. Some rules must nonetheless be fixed to have an implementation.

The class Smallest Ring consists of a molecule that is an aromatic ring, associated to a particular aromatic atom. This ring is the smallest aromatic ring the atom belongs to. If it belongs to several rings of same size, the choice is dependant on the memory address allocated to the atom. It could depend on the OpenEye library, the compiler and the computer.

9.2 Smallest Ring application program interface

9.2.1 Reset

Reset the ring. The ring molecule is empty.

```
void Reset()
```

9.2.2 GetSmlstRng

Get the ring molecule associated to the object.

```
inline OEMol GetSmlstRng()
```

9.2.3 GetSeekId

Get the aromatic atom index for which the ring is searched.

```
inline int GetSeekId()
```

9.2.4 GetMaxCnt

Search the maximum ring size the aromatic atom can belong to.

```
inline int GetMaxCnt()
```

9.2.5 SetSeekId

Set the index of the aromatic atom for which the ring is searched.

```
inline void SetSeekId(const int &i)
```

9.2.6 SetMaxCnt

Set the maximum ring size the aromatic atom can belong to.

```
inline void SetMaxCnt(const int &i)
```

9.2.7 SetSearch

Set for a particular aromatic atom the search of the ring.

```
void SetSearch(OEAtomBase &atom)
```

9.2.8 FndARng

Find the ring.

```
bool FndARng(OEAtomBase *atom, unsigned int PrevId); //Find the ring
```

9.2.9 CleanRng

In the ring molecule resulting from the search, the first and the last atom are the same. This function cleans this redundancy.

```
void CleanRng()
```


Chapter 10

Medoid tree data organization

Organizing data in clusters can be done in very numerous ways once relationships between individuals elements are found. A practical way to cluster data is to find a hierarchical organization such as trees.

10.1 Presentation and concepts

This library is based on the C cluster library¹. This library gives a number of functions to generate different kinds of clusters and trees from a set of data. The methods could be roughly divided in metric and distance methods. Though these concepts are related, a metric-method clustering function is based on a set of data for which a rule is given to compare them, while distance-based clustering methods use a distance matrix based on the data.

The tree methods provided are distance-based. They agglomerate the data by pairs of first neighbors. Another distance-based clustering method is termed k -medoid. It consists of dividing the datas in k subsets organized around k elements, called medoids. These elements are chosen randomly. An element belongs to the cluster from which medoid it is the closest. The process is repeated a number of times until a consensus is chosen.

This methods allows a simple procedure to divide the data in a hierarchical tree. As any tree is equivalent to a binary tree, binary division of the data at each steps is sufficient. So the data and the related distance matrix are recursively divided in 2-medoids clusters.

¹<http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/>

10.2 Tree medoid application program interface

This is a library of function not a class. This means that these functions do not define an object.

10.2.1 DistNode

Get the distance between the first two nodes of the clustid array found to be non-null according to the distmatrix distance matrix.

```
double DistNode(int clustid[],double** distmatrix);
```

10.2.2 MatrixDivide

Use as input the ncluster clusters, the rank of the distance matrix, the clusturid array referencing the cluster membership, the OldId and NewId array referencing the original and current indexing of the datas, the distMatrix distance matrix. The result is a kMatrix vector of distance matrix, of ranks listed int the ikmax array, that is the division of the original distance matrix on the ncluster new clusters.

```
void MatrixDivide(int ncluster, int rank,
                  int clusterid[],
                  int OldId[],vector<int> NewId[],
                  double** distMatrix,
                  vector<double**> &kMatrix,
                  int ikmax[])
```

10.2.3 treekmedoid

This function allows to perform the 2-medoid recursive tree construction from the data. The result is a bidimensional array listing the connectivity of the binary tree and an array of the linked distances. It uses the following parameters:

- rank: the rank of the distance matrix.
- npass: the number of times the 2-medoid must be repeated until a consensus is found.
- distMatrix: triangular distance matrix.
- OldId: array of the index of the elements to be clustered.
- dist_old: used only for recursivity. Does not need to be set.
- result: bi-dimensional array of the connectivity of the tree.
- linkdist: distance between tree nodes.

The data structures conform exactly to the one of the C cluster library. Some help can be found in the example program of the C cluster library for precise examples of how to construct the input distance matrix.

This particular design was originally designed by the authors of the C cluster library to optimize memory efficiency. This is why it is pointless to design a simpler interface: the memory cost of such interface would destroy the original design.

```
void treemedoid(int rank,int npass,double** distMatrix,
               int OldId[],double dist_old,
               int result[][2],double linkdist[])
```

10.3 Cluster and Nexus formatted trees application program interface

This library manages the relationship between output tree formats, Nexus, and the inner representation of the tree as hierarchical clusters. This inner representation is conform to the specifications of the C cluster library.

10.4 NexusTreeOut

This function converts the output of C cluster libraries that create trees into a string that represents it in the Nexus format. It uses the following parameters:

- nrows: the number of clustered elements.
- result: bi-dimensional array of the connectivity of the tree.
- linkdist: distance between tree nodes.
- OutName: if present, the file name to print the result.
- ListEle: the label of each clustered element.

```
void NexusTreeOut(int nrows, int result[][2],
                  double linkdist[], string OutName, vector<string> ListEle){
void NexusTreeOut(int, int result[][2], double linkdist[], vector<string>);
```

If OutName is not specified the output is the standard output stdout.

10.5 NexusFileIn

This function reads in a tree in Nexus format to extract information in order to convert it in a C cluster libraries compliant form.

- NodeNames: a vector of string that stores the names of the nodes..
- NexusFileName: a file containing a tree in Nexus format.
- Nexus: a string storing the tree.

```
void NexusFileIn(vector<string> &NodeNames, string NexusFileName,
                 string &Nexus){
```

10.6 NexusTreeIn

This function converts a string representing Nexus formatted tree into a C cluster library compliant form.

- Nexus: a string storing the tree.
- results: a bidimensional array, containing the tree topology
- distance: 1D array representation of the symmetric distance matrix.
- nrows: integer specifying the size of the tree. Same as in the C cluster library.

```
void NexusTreeIn(string &Nexus, int results[][2], double distance[],  
                int &nrows)
```

Chapter 11

Machine learning: Bayesian classifier

This chapter describes the Bayes class. This class should change structure as soon as possible to become more generic and allow for development of a diversity of machine learning algorithms, including Bayes equation, SVM, neuronal networks, etc. The current implementation must be understood as a prototype.

11.1 Presentation and concepts

The purpose of the Bayes class is to be the basis of machine learning algorithms. The boolean problem consists of assessing the activity or not of a compound given its fingerprint. Other formulations shall be considered in the future.

Two approaches are implemented. The first one is based on the Bayes equation. The second one, following the “naive Bayes algorithm” implemented in Pipeline Pilot¹, is based on a discriminant function of Xia et al ².

¹SciTegic Inc.

²*J. Med. Chem.*, 2004, **47**, 4463-4470

11.2 Bayes application program interface

Here are described the different methods of this class.

11.2.1 Constructor

Create a learning machine and initialize it.

```
Bayes()
```

11.2.2 Reset

Reinitialize class variables.

```
void Reset()
```

11.2.3 SetAlgo

Set the algorithm flag. This flag is an integer. A value of 0 stands for a Bayesian learning machine, a value of 1 stands for a Xia's discriminant function learning machine.

```
inline void SetAlgo(int i)
```

11.2.4 GetAlgo

Get the algorithm flag.

```
inline int GetAlgo()
```

11.2.5 Training

Train the learning machine with a descriptor vector and a boolean flag indicating if the corresponding bit vector stand for an active one or an inactive one.

```
void Training(OEBitVector, bool)
```

11.2.6 SetInfereXia

Infer the Xia discriminant function for each descriptor.

The Xia's function is based on the number of actives in the training set containing the descriptor A_{H_i} , the total number of members of the training set containing the descriptor T_{H_i} , the total number of active A and the size of the training set T :

$$\chi_i = \left(\frac{A_{H_i} + K \frac{A}{T}}{T_{H_i} + K} \right) \left(\frac{T}{A} \right) \quad (11.1)$$

with K a factor controlling the meaningfulness of a descriptor compared to the statistical sampling of the descriptor. Note that when $K \rightarrow 0$, $\chi_i \rightarrow \frac{A_{H_i} T}{T_{H_i} A}$ that is an estimator of the activity probability knowing that the descriptor H_i is on, divided by an estimator of the activity. So when K is small this means that the Xia's discriminant function is confident in the probability estimators. On another hand when K is big, $\chi_i \rightarrow K$, so that the behavior of the function reflects no more the training set, usually because it is not statistically significant for that particular descriptor.

The current implementation of Xia's function uses $K = \frac{T}{A}$ and takes the logarithm. For bits on:

$$\log(\chi_{H_i}) = \log\left(\frac{TA_{H_i} + T}{AT_{H_i} + T}\right) \quad (11.2)$$

For bits off:

$$\log(\chi_{\bar{H}_i}) = \log\left(\frac{TA_{\bar{H}_i} + T}{AT_{\bar{H}_i} + T}\right) \quad (11.3)$$

The log is advantageous since it is positive for an "increase of probability" compared to the *a priori* activity probability and negative otherwise. The total score is then a simple sum.

$$\log(\chi(\{H_i\}, \{\bar{H}_j\})) = \sum_i \log(\chi_{H_i}) + \sum_j \log(\chi_{\bar{H}_j}) \quad (11.4)$$

The sum is carried over all bits. When a bit is observed on the 11.2 is added, when it is observed off the 11.3 is added.

```
void SetInfereXia()
```


11.2.7 SetInferBayes

Infer the Bayes conditional probability of activity knowing a descriptor is on.

We estimate the probability of having a descriptor knowing the activity or inactivity as:

$$P(active|H_i) = \frac{A_{H_i}}{A_{H_i} + \bar{A}_{H_i}} \quad (11.5)$$

$$P(active|\bar{H}_i) = \frac{A_{\bar{H}_i}}{A_{\bar{H}_i} + \bar{A}_{\bar{H}_i}} \quad (11.6)$$

with A being the number of actives, \bar{A} the number of inactives, A_{H_i} the number of active with the descriptor on, \bar{A}_{H_i} the number of inactive with the descriptor on, $A_{\bar{H}_i}$ the number of active with the descriptor off, $\bar{A}_{\bar{H}_i}$ the number of inactive with the descriptor off.

The naive assumption of uncorrelated descriptors then relate the total probability to a product:

$$P(active|\{H_i\}\{\bar{H}_j\}) = \prod_i P(active|H_i) \prod_j P(active|\bar{H}_j) \quad (11.7)$$

The product is carried over all statistically relevant bits. When a bit is observed on the 11.5 is used, when it is observed off the 11.6 is used.

To decide whether a bit is relevant or not, its on and off support value must be above a threshold. See the following sections (11.2.10, 11.2.13, 11.2.14).

Note: if a descriptor was never met on, its on contribution is 0. The same choice is made for the off contribution when a descriptor was never met off.

Note: We took the logarithm of this conditional probability because of numerical stability. The products in 11.7 are transformed in sums. We set the logarithm of a 0 probability to -10000 .

```
void SetInferBayes()
```

11.2.8 SetOnSupport

Set the `OnSupport` double vector.

```
inline void SetOnSupport(vector<double> Obs)
```

11.2.9 SetOffSupport

Set the OffSupport double vector.

```
inline void SetOffSupport(vector<double> Obs)
```

11.2.10 SetSupport

Set the OnSupport and OffSupport double vector. Those vectors carry an estimated support value for each descriptor. The on support is the probability to observe an active with the descriptor on ($P(A \cap H_i)$) and the off support is the probability to observe an active with the descriptor off ($P(A \cap \bar{H}_j)$). These values describe the relevance of the associated descriptor. If a strong correlation is observed between a descriptor and the activity property, this is meaningless if at the same time the probability to observe both at the same time is weak.

Considering that A_{H_i} is the number of actives with the descriptor i on, that $A_{\bar{H}_j}$ is the number of actives with the descriptor j off and T is the total size of the training set, the support value is estimated as:

$$P(A \cap H_i) = \frac{A_{H_i}}{T} \quad (11.8)$$

$$P(A \cap \bar{H}_j) = \frac{A_{\bar{H}_j}}{T} \quad (11.9)$$

$$(11.10)$$

This prevents to use insufficiently sampled descriptors to estimate the conditional probability of activity knowing the feature.

```
inline void SetSupport()
```

11.2.11 GetOnSupport

Get the OnSupport double vector.

```
inline vector<double> GetOnSupport()
```

11.2.12 GetOffSupport

Get the OffSupport double vector.

```
inline vector<double> GetOffSupport()
```

11.2.13 SetOnSupportMin

Set the minimal value for the support value of an on state descriptor to be relevant.

```
inline void SetOnSupportMin(double m)
```

11.2.14 SetOffSupportMin

Set the minimal value for the support value of an off state descriptor to be relevant.

```
inline void SetOffSupportMin(double m)
```

11.2.15 GetOnSupportMin

Get the minimal value for the support value of an on state descriptor to be relevant.

```
inline double GetOnSupportMin()
```

11.2.16 GetOffSupportMin

Get the minimal value for the support value of an off state descriptor to be relevant.

```
inline double GetOffSupportMin()
```

11.2.17 SetOnInfer

Set the infer values when an on state is observed for the descriptors.

```
inline void SetOnInfer(vector<double> VI)
```

11.2.18 SetOffInfer

Set the infer values when an off state is observed for the descriptors.

```
inline void SetOffInfer(vector<double> VI)
```

11.2.19 GetOnInfer

Get the infer values when an on state is observed for the descriptors.

```
inline vector<double> GetOnInfer()
```

11.2.20 GetOffInfer

Get the infer values when an off state is observed for the descriptors.

```
inline vector<double> GetOffInfer()
```

11.2.21 Predict

Predict the value of a discriminant function or the conditional probability given a complete set of descriptors.

```
double Predict(OEBitVector);
```

Chapter 12

Machine learning: SVM

This chapter describes the SVM class. This class is a C++ wrapper -a bit incomplete- for libsvm 2.8.

12.1 Presentation and concepts

The SVM trains itself on a two class problem: inactives versus actives. Each subsequent unknown ligand-protein interaction fingerprint will be compared to those of the training set to build a vector of tanimoto scores. These vectors will be used as support vectors for the SVM.

The SVM class is defined in the files `SVMBV.hpp` and `SVMBV.cc`.

12.2 SVM application program interface

Here are described the different methods of this class.

12.2.1 Constructor

Build an SVM. The data structure includes a training set, parameters of the SVM and a model.

```
SVM();
```

12.2.2 Destructor

Delete the SVM.

```
~SVM();
```

12.2.3 Reset

Reset the data structure of the SVM.

```
void Reset();
```

12.2.4 AddBVTrain

Add a bit vector to the training set.

```
void AddBVTrain(OEBitVector BV)
```

12.2.5 SetBVTrain

Set the bit vector training set.

```
void SetBVTrain(vector<OEBitVector> BV)
```

12.2.6 GetBVTrain

Get the bit vector training set.

```
vector<OEBitVector> GetBVTrain()
```

12.2.7 AddClasses

Add a class to the training set. It is the label (active or inactive) of the bit vector sharing the same index.

```
void AddClasses(double d)
```

12.2.8 SetClasses

Set classes to the training set. It is the label (active or inactive) of the bit vector sharing the same index.

```
inline void SetClasses(vector<double> d)
```

12.2.9 GetClasses

Get classes of the training set.

```
vector<double> GetClasses()
```

12.2.10 SetSVMProblem

Build the SVM problem data structure.

```
void SetSVMProblem(svm_problem prb)
```

12.2.11 GetSVMProblem

Get the SVM problem data structure.

```
svm_problem GetSVMProblem()
```

12.2.12 SetSVMParameter

Set the SVM parameters.

```
void SetSVMParameter(svm_parameter prm)
```

12.2.13 GetSVMParameter

Get the SVM parameters.

```
svm_parameter GetSVMParameter()
```


12.2.14 Training

Set the model. Use the problem data structure to train the SVM.

```
void Training(vector<OEBitVector>&, vector<double>&);
```

12.2.15 SetModel

Set the model.

```
void SetModel();
```

12.2.16 GetModel

Get the model.

```
inline svm_model* GetModel(){
```

12.2.17 Predict

Use the model to predict classes of unknown ligand-protein interaction fingerprints.

```
double Predict(OEBitVector);
```

Chapter 13

Installation

The FingerPrintLib is build around OpenEye and Boost libraries. Cluseter2TreeView, treemedoid and cluster using applications make use also of the C cluster library. SVMbV applications need the libsvm library.

13.1 Boost library

This library can be downloaded free of charges at this address:

<http://www.boost.org/index.htm>

Installation instructions can be found there:

http://www.boost.org/more/getting_started.html

13.1.1 Boost license

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in

the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

13.1.2 Where are Boost libraries being used?

In fact only the `program_options` and `regex` library are used. The library `program_options` allowed to develop the necessarily complex user interface for these programs in a readable, fast to implement and easy to maintain form. The `regex` library was useful to implement Nexus tree format support.

It is not necessary to compile all the Boost library but only the `program_options` and `regex` ones. To do so, in step 5 of the installation, use for example, the command:

```
bjam --with-libboost\_program\_options -sTOOLS=gcc
bjam --with-libboost\_regex -sTOOLS=gcc
```

without forgetting to adapt this command to your particular system.

Nonetheless, Boost are very powerful libraries. Complete installation of the libraries is highly recommended since it will provide tools for many much more than those particular applications.

13.2 OpenEye

The OpenEye libraries can be downloaded at this site:

<http://www.eyesopen.com/download/#OECHEM>

These libraries are proprietary. Licenses for academics are free of charge. See this page for details:

http://www.eyesopen.com/forms/academic_license_app.php

13.2.1 Where are OpenEye libraries being used?

The libraries are used at several points:

- interfacing with almost all common molecular formats
- managing atoms and molecules (index, type, properties, linkage)
- basic mathematics (punctual space calculations)
- bit vector manipulation (Tanimotos calculations).
- SMARTS support.

OpenEye libraries are far more extended than OEChem, and are really worth the money asked.

13.2.2 OpenEye and OpenBabel

OpenEye released a few years ago the OELib libraries that were a C++ port of Babel. OpenEye decided later to rewrite a better set of libraries that finally became OEChem. OpenBabel take off were OELib left. OpenBabel is Open Source and is distributed under GPL.

For this particular application nonetheless, very little work could be necessary to use only OpenBabel libraries. OpenBabel libraries can be downloaded free of charges at this site:

<http://openbabel.sourceforge.net/>

Some information to perform this migration can be found at this site:

<http://openbabel.sourceforge.net/Migration.shtml>

13.3 C Clustering Library

This library is used by the FingerPrint application. Its goal is to provide a rich set of clustering methods. The library as well as many interesting applications and extensions can be downloaded free of charge at this site:

<http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/software.htm>

13.3.1 License

A. HISTORY OF THE SOFTWARE

=====

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI) in the Netherlands as a successor of a language called ABC. Guido is Python's principal author, although it includes many contributions from others. The last version released from CWI was Python 1.2. In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI) in Reston, Virginia where he released several versions of the software. Python 1.6 was the last of the versions released by CNRI. In 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. Python 2.0 was the first and only release from BeOpen.com.

Following the release of Python 1.6, and after Guido van Rossum left CNRI to work with commercial software developers, it became clear that the ability to use Python with software available under the GNU Public License (GPL) was very desirable. CNRI and the Free Software Foundation (FSF) interacted to develop enabling wording changes to the Python license. Python 1.6.1 is essentially the same as Python 1.6, with a few minor bug fixes, and with a different license that enables later versions to be GPL-compatible. Python 2.0.1 is a derivative work of Python 1.6.1, as well as of Python 2.0.

After Python 2.0 was released by BeOpen.com, Guido van Rossum and the other PythonLabs developers joined Digital Creations. All intellectual property added from this point on, including Python 2.0.1 and its alpha and beta releases, is owned by the Python Software Foundation (PSF), a non-profit modeled after the Apache Software Foundation. See <http://www.python.org/psf/> for more information about the PSF.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON
=====

PSF LICENSE AGREEMENT

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.0.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.0.1 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001 Python Software Foundation; All Rights Reserved" are retained in Python 2.0.1 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.0.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.0.1.

4. PSF is making Python 2.0.1 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.0.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.0.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.0.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python 2.0.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM TERMS AND CONDITIONS FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS

FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI OPEN SOURCE GPL-COMPATIBLE LICENSE AGREEMENT

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide

license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright (c) 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>".

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without

limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI PERMISSIONS STATEMENT AND DISCLAIMER

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO

THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

13.4 libSVM

This library is used by the SVM application. Its goal is to provide a rich set of SVM methods. The library as well as many interesting applications and extensions can be downloaded free of charge at this site:

<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

13.4.1 License

Copyright (c) 2000-2006 Chih-Chung Chang and Chih-Jen Lin
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither name of copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
“AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

13.5 Application installation

To install IFP, IAFP, MyIFP, LBMFP, Bayes, SVM or T2C, first install the preceeding libraries.

Then modify the Makefiles such that the variable OEDIR matches your OpenEye libraries location. Modify the variable INCBOOST to match the location of the Boost libraries headers. Modify the variable INCSVM and OBJSVMS to match your local installation of libsvm.

Copy from your installation of the C clustering library to the source directory of these applications, the following files:

- cluster.h
- cluster.c
- the directory ranlib

For LBMFP, type `make -f Makefile.LBMFP`.

For ExtLBMFP, type `make -f Makefile.ExtLBMFP`.

For IFP, type `make -f Makefile.IFP`.

For ExtIFP, type `make -f Makefile.ExtIFP`.

For IAFP, type `make -f Makefile.IAFP`.

For MyIFP, type `make -f Makefile.MyIFP`.

For Bayes, type `make -f Makefile.Bayes`.

For SVM, type `make -f Makefile.SVM`.

For T2C, type `make -f Makefile.T2C`.

Objects will be created in the `/FingerPrintLib/obj` directory and binaries will be stored in the `/FingerPrintLib/bin` directory. That's all.

Chapter 14

Application manual

Seven applications were derived from the preceeding libraries.

1. IFP clusters docked poses in a binding site
2. IAFP clusters docked poses using ligand-focused fingerprints
3. MyIAFP clusters docked poses using limited SMARTS supported fingerprints
4. LBMFP compares docked poses to some reference poses in the same binding site and outputs a similarity tanimoto score.
5. Bayes uses a set of active and inactive compounds to train a Bayesian machine learning to recognize the fingerprints of actives
6. SVM trains an SVM machine learning to recognize fingerprints of actives
7. T2C allows one to read a Nexus tree file and split the tree into the desired number of clusters

14.1 IFP, IAFP, MyIFP, ExtIFP

IFP and MyIFP programs take a set of amino acids describing the cavity of the protein and compare contacts formed by docked poses to that cavity. These contacts are described by bit vectors. Each docked pose having one bit vector describing it, similarity between them can be evaluated by a Tanimoto score that provide a distance between each pose relative to each other. The

resulting matrix can then be used to generate clusters of the docked poses based on the binding mode.

14.1.1 Command line

IFP

The command line is:

```
IFP [-options] -C <Cavity.mol2> -L <List of docked poses file>
MyIFP [-options] -C <Cavity.mol2> -L <List of docked poses file>
ExtIFP [-options] -C <Cavity.mol2> -L <List of docked poses file>
```

Allowed options are:

- -h [-help] : produces help message
- -O [-Outname] : Output file/tree name [Default=usr.tre]
- -m [-ClustMeth] : Hierarchical clustering method (s|m|a|k) [Default=s]
- -A [-LAro] : Ring centered aromatic interaction [Default=false]
- -HbndLngth : Hbond length (Angström) [Default=2.8]
- -HbndAngle : Hbond angle (rad) [Default=0.0]
- -HbndDngle : Hbond angle tolerance (rad) ([Default=1.047]
- -FFLngth : Aromatic face to face interaction length (Angstrom) [Default=4.0]
- -FFAngle : Aromatic face to face interaction angle (rad) [Default=0.0]
- -FFDngle : Aromatic face to face interaction angle tolerance (rad) ([Default=1.047]
- -EFLngth : Aromatic edge to face interaction length (Angstrom) [Default=4.0]
- -EFAngle : Aromatic edge to face interaction angle (rad) [Default=1.57]

- `-EFDngle` : Aromatic edge to face interaction angle tolerance (rad) [Default=1.047]
- `-HyLngth` : Hydrophobic interaction length (Angström) [Default=3.0]
- `-IoLngth` : Ionic interaction length (Angström) [Default=4.0]

the file of docked poses must contain the name of the files (full paths can be accepted) containing the docked poses. Each line should contain only one file name. Each docked pose file should contain only one pose.

The `-m` switch for selecting the clustering method support four different values:

- `s` pairwise single-linkage clustering
- `m` pairwise maximum- (or complete-) linkage clustering
- `a` pairwise average-linkage clustering
- `k` recursive 2-medoid hierarchical clustering

ExtIFP

The ExtIFP program has more options:

- `-WHbndLngth arg (=2.5)` : Weak Hbond length (Angström) [Default= 3.50]
- `-WHbndAngle arg (=0)` : Weak Hbond angle (rad) [Default= 0.00]
- `-WHbndDngle arg (=0.52)` : Weak Hbond angle tolerance (rad) ([Default= 0.52]
- `-PCLngth arg (=4)` : Pi-Cation length (Angström) [Default= 3.50]
- `-PCAngle arg (=0)` : Pi-Cation angle (rad) [Default= 0.00]
- `-PCDngle arg (=0.52)` : Pi-Cation angle tolerance (rad) ([Default= 0.52]
- `-MeLngth arg (=2)` : Metal interaction length (Angström) [Default= 2.00]

MyIFP

The program MyIFP expects the presence of a one blank separated line file `Smarts.txt` containing two SMARTS and a real. They describe the SMARTS to be recognized and the maximum distance between them so they are considered interacting with each other. Currently the interaction center for each atom in a part of a molecule mapped by a SMARTS is the center of the atom. So it is the shortest distance between the two SMARTS group that will be compared to the interaction threshold.

IAFP

The program IAFP has a slightly different interface:

```
IAFP [-options] -L <column 1:ligand files names  
                    column 2: cavity files names>
```

The same ligand adopts different conformations in different cavities. So the input file has to give the ligand conformer file name and its corresponding cavity file name.

Allowed options are the same as for the two other applications.

14.1.2 Command file

If a file IFP.cmd exists, it will be read to change default value options. The extended name of the option is expected. That is if someone does not want to distinguish between face-to-face and edge-to-face interactions, one can create a file IFP with the following content:

```
EFLngth=0  
FFDngle=6.29  
Outname=MyTree.tre
```

This will set the edge to face interaction to 0 so no such kind of interactions could no longer be recorded and it set the tolerance for face-to-face interaction to 2π , so that any relative orientation of aromatic ring to one another will be accepted. The last line, tell to save the resulting tree, in a file called MyTree.tre.

14.2 LBMFP,ExtLBMFP

LBMFP is a program that takes a set of amino acids describing the cavity of the protein. It compares contacts formed by docked poses to that cavity to those contacts formed by reference poses in the same cavity. These contacts are described by bit vectors. Each docked pose having one bit vector describing it, similarity between them can be evaluated by Tanimoto scores that provide a distance between each pose relative to the reference poses. The result is set of Tanimoto scores, one for each docked ligand and reference pose.

ExtLBMFP uses extended version of the intermolecular interaction fingerprints. As a consequence, the bit vectors are longer.

14.2.1 Command Line

LBMFP

The command line is:

```
LBMFP [-options] -C <Cavity.mol2> -L <poses file List> -R <Reference list>
ExtLBMFP [-options] -C <Cavity.mol2> -L <poses file List> -R <Reference list>
```

Allowed options are:

- -h [-help] : produces help message
- -O [-Outname] arg (=usr.tre) : Output file/tree name [Default=usr.tre]
- -B [-BitVector] : References are bit vectors [Default=false]
- -A [-LAro] : Ring centered aromatic interaction [Default=false]
- -HbndLngth arg (=2.8) : Hbond length (Angström) [Default=2.8]
- -HbndAngle arg (=0) : Hbond angle (rad) [Default=0.0]
- -HbndDngle arg (=1.047) : Hbond angle tolerance (rad) ([Default=1.047]
- -FFLngth arg (=4) : Aromatic face to face interaction length (Angström) [Default=4.0]

- -FFAngle arg (=0) : Aromatic face to face interaction angle (rad) [Default=0.0]
- -FFDngle arg (=1.047) : Aromatic face to face interaction angle tolerance (rad) ([Default=1.047]
- -EFLngth arg (=4) : Aromatic edge to face interaction length (Angström) [Default=4.0]
- -EFAngle arg (=1.57) : Aromatic edge to face interaction angle (rad) [Default=1.57]
- -EFDngle arg (=1.047) : Aromatic edge to face interaction angle tolerance (rad)[Default=1.047]
- -HyLngth arg (=4) : Hydrophobic interaction length (Angström) [Default=4.0]
- -IoLngth arg (=4) : Ionic interaction length (Angstrom) [Default=4.0]

the files of docked poses must contain the name of the files containing the docked poses. Each line should contain only one file name. Each docked pose file may contain more than one pose. It is recommended that the title of the poses contains some numbering since this name is the one that will be saved in the output. So if several poses of the same molecule has the same title, the output scores will not be easily assignable to a particular pose.

The references can be a set of docked pose files. Alternatively, if the option -B is present, the file will be interpreted as a set of reference bit vectors. Each line of the file must contain a name (without blanks) for the reference and a set of contiguous 0 and 1 describing the reference bit vector fingerprint. The length and the actual adequation of the input bit vector with the input cavity is the responsibility of the user. No error or warning message will be displayed if there is some inconsistencies.

ExtLBMFP

The ExtLBMFP program has more options:

- -WHbndLngth arg (=2.5) : Weak Hbond length (Angstrom) [Default=3.50]

- `-WHbndAngle arg (=0)` : Weak Hbond angle (rad) [Default= 0.00]
- `-WHbndDngle arg (=0.52)` : Weak Hbond angle tolerance (rad) ([Default= 0.52]
- `-PCLngth arg (=4)` : Pi-Cation length (Angstrom) [Default= 3.50]
- `-PCAngle arg (=0)` : Pi-Cation angle (rad) [Default= 0.00]
- `-PCDngle arg (=0.52)` : Pi-Cation angle tolerance (rad) ([Default= 0.52]
- `-MeLngth arg (=2)` : Metal interaction length (Angstrom) [Default= 2.00]

14.2.2 Command file

If a file IFP.cmd exists, it will be read to change default value options. The extended name of the option is expected. That is if someone does not want to distinguish between face-to-face and edge-to-face interactions, one can create a file IFP with the following content:

```
EFLngth=0
FFDngle=6.29
Outname=MyTree.tre
```

This will set the edge to face interaction to 0 so no such kind of interactions could no longer be recorded and it set the tolerance for face-to-face interaction to 2π , so that any relative orientation of aromatic ring to one another will be accepted. The last line, tell to save the resulting tree, in a file called MyTree.tre.

14.3 Bayes

Bayes is a program that takes a set of amino acids describing the cavity of the protein. It compares contacts formed by docked poses to that cavity to those contacts formed by reference poses in the same cavity.

A set of reference poses or fingerprints are given with a boolean flag indicating for each if it describes an active or an inactive. Those are used to train

a learning machine. This machine can then be used to infer a discriminant function or a conditional probability that an unknown finger print correspond to an active or an inactive one.

The result is a set of probability or a set of scores.

14.3.1 Command Line

The command line is:

```
Bayes [-options] -C <Cavity.mol2> -L <poses file List> -R <Reference list>
```

Allowed options are:

- -h [-help] : produces help message
- -C [-Cavity] arg : Cavity file name
- -R [-RefList] arg : List of references file name (one per line)
- -L [-Liglist] arg : List of ligand file name (one per line)
- -O [-Outname] arg (=Out.bay) : Output file/tree name [Default=Out.tan]
- -B [-BitVector] : References are bit vectors [Default=false]
- -A [-LAro] : Ring center aromatic interaction [Default=false]
- -M [-Machine] arg (=1) : Kind of Bayesian learning machine (0|1) [default=1]
- -OnMin arg (=0.1) : Minimal support value for a bit on to be significant [default=0.1]
- -OffMin arg (=0.1) : Minimal support value for a bit off to be significant [default=0.1]
- -HbndLngth arg (=3.5) : Hbond length (Angström) [Default= 3.50]
- -HbndAngle arg (=0) : Hbond angle (rad) [Default= 0.00]
- -HbndDngle arg (=0.5235987755982988) : Hbond angle tolerance (rad) ([Default= 0.52]

- `-FFLength arg (=4.5)` : Aromatic face to face interaction length (Angström) [Default= 4.50]
- `-FFAngle arg (=0)` : Aromatic face to face interaction angle (rad) [Default= 0.00]
- `-FFDngle arg (=0.5235987755982988)` : Aromatic face to face interaction angle tolerance (rad) [Default= 0.52]
- `-EFLength arg (=3.5)` : Aromatic edge to face interaction length (Angström) [Default= 3.50]
- `-EFAngle arg (=0.5235987755982988)` : Aromatic edge to face interaction angle (rad) [Default= 0.52]
- `-EFDngle arg (=0.5235987755982988)` : Aromatic edge to face interaction angle tolerance (rad) [Default= 0.52]
- `-HyLength arg (=3.5)` : Hydrophobic interaction length (Angström) [Default= 3.50]
- `-IoLength arg (=4)` : Ionic interaction length (Angström) [Default= 4.00]

the files of reference docked poses must contain the name of the files containing the docked poses followed by a 0 for an inactive and a 1 for an active. Each line should contain only one file name.

Alternatively, if the option `-B` is present, the file will be interpreted as a set of reference bit vectors. Each line of the file must contain a name (without blanks) for the reference and a set of contiguous 0 and 1 describing the reference bit vector finger print and the activity flag. The length and the actual adequation of the input bit vector with the input cavity is the responsibility of the user. No error or warning message will be displayed if there is some inconsistencies.

The `-M` switch control the algorithm in use:

- 0 Bayesian learning machine
- 1 Xia's discriminant function, as in Pipeline Pilot

The on and off minimal support values are irrelevant for Xia's discriminant function. If a descriptor was never met during the training, the contribution to the score is null.

14.3.2 Command file

If a file IFP.cmd exists, it will be read to change default value options. The extended name of the option is expected. That is if someone does not want to distinguish between face-to-face and edge-to-face interactions, one can create a file IFP with the following content:

```
EFLngth=0  
FFDngle=6.29  
Outname=MyTree.tre
```

This will set the edge to face interaction to 0 so no such kind of interactions could no longer be recorded and it set the tolerance for face-to-face interaction to 2π , so that any relative orientation of aromatic ring to one another will be accepted. The last line, tell to save the resulting tree, in a file called MyTree.tre.

14.4 SVM

SVM is a program that takes a set of amino acids describing the cavity of the protein. It compares contacts formed by docked poses to that cavity to those contacts formed by reference poses in the same cavity.

A set of reference poses or fingerprints are given with a real number flag indicating for each, a degree of activity. Those are used to train an SVM learning machine. This machine can then be used to infer an activity for an unknown fingerprint.

The result is a set of scores.

14.4.1 Command Line

The command line is:

```
SVM [-options] -C <Cavity.mol2> -L <poses file List> -R <Reference list>
```

Allowed options are:

- -h [-help] : produces help message

- -C [-Cavity] arg : Cavity file name
- -R [-RefList] arg : List of references file name (one per line)
- -L [-Liglist] arg : List of ligand file name (one per line)
- -O [-Outname] arg (=Out.bay) : Output file/tree name [Default=Out.tan]
- -B [-BitVector] : References are bit vectors [Default=false]
- -s [-svm_type] arg (=0) : set type of SVM [Default 0]
- -t [-kernel_type] arg (=2) : set type of kernel function [Default 2]
- -d [-degree] arg (=3) : set degree in kernel function [Default 3]
- -g [-gamma] arg (=1) : set gamma in kernel function [Default 1.0]
- -r [-coef0] arg (=0) : set coef0 in kernel function [Default 0.0]
- -c [-cost] arg (=1) : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR [Default 1.0]
- -n [-nu] arg (=0.5) : nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR [Default 0.5]
- -p [-svr_epsilon] arg (=0.1) : set the epsilon in loss function of epsilon-SVR [Default 0.1]
- -m [-cachesize] arg (=100) : set cache memory size in MB [Default 100.0]
- -e [-epsilon] arg (=0.001) : set tolerance of termination criterion [Default 0.001]
- -shrinking arg (=1) : whether to use the shrinking heuristics, 0 or 1 [Default 1]
- -probability_estimates arg (=1) : whether to train a SVC or SVR model for probability estimates, 0 or 1 [Default 0]
- -w [-weight] arg (=1) : set the parameter C of class i to weight*C, for C-SVC [Default 1]

- -v [-n_fold] arg (=0) : n-fold cross validation mode [Default 0]. NOT SUPPORTED.
- -A [-LAro] : Ring center aromatic interaction [Default=false]
- -HbndLngth arg (=3.5) : Hbond length (Angström) [Default= 3.50]
- -HbndAngle arg (=0) : Hbond angle (rad) [Default= 0.00]
- -HbndDngle arg (=0.5235987755982988) : Hbond angle tolerance (rad) ([Default= 0.52]
- -FFLngth arg (=4.5) : Aromatic face to face interaction length (Angström) [Default= 4.50]
- -FFAngle arg (=0) : Aromatic face to face interaction angle (rad) [Default= 0.00]
- -FFDngle arg (=0.5235987755982988) : Aromatic face to face interaction angle tolerance (rad) ([Default= 0.52]
- -EFLngth arg (=3.5) : Aromatic edge to face interaction length (Angström) [Default= 3.50]
- -EFAngle arg (=0.5235987755982988) : Aromatic edge to face interaction angle (rad) [Default= 0.52]
- -EFDngle arg (=0.5235987755982988) : Aromatic edge to face interaction angle tolerance (rad)[Default= 0.52]
- -HyLngth arg (=3.5) : Hydrophobic interaction length (Angström) [Default= 3.50]
- -IoLngth arg (=4) : Ionic interaction length (Angström) [Default= 4.00]

the files of reference docked poses must contain the name of the files containing the docked poses followed by a real number describing the degree of activity. It can also be 0 for an inactive and 1 for an active. Each line should contain only one file name.

Alternatively, if the option -B is present, the file will be interpreted as a set of reference bit vectors. Each line of the file must contain a name

(without blanks) for the reference and a set of contiguous 0 and 1 describing the reference bit vector finger print and the activity flag. The length and the actual adequation of the input bit vector with the input cavity is the responsibility of the user. No error or warning message will be displayed if there is some inconsistencies.

The -s switch control the SVM type in use:

- 0 C-SVC
- 1 ν -SVC
- 2 one-class SVM
- 3 ϵ -SVR
- 4 ν -SVR

The -t switch control the SVM kernel in use:

- 0 linear: $\langle u|v \rangle$
- 1 polynomial: $(\gamma \langle u|v \rangle + \text{coef}_0)^{\text{degree}}$
- 2 radial basis function: $e^{-\gamma * ||u-v||^2}$
- 3 sigmoid: $\tanh(\gamma \langle u|v \rangle + \text{coef}_0)$

14.4.2 Command file

If a file IFP.cmd exists, it will be read to change default value options. The extended name of the option is expected. That is if someone does not want to distinguish between face-to-face and edge-to-face interactions, one can create a file IFP with the following content:

```
EFLngth=0
FFDngle=6.29
Outname=MyTree.tre
```

This will set the edge to face interaction to 0 so no such kind of interactions could no longer be recorded and it set the tolerance for face-to-face interaction to 2π , so that any relative orientation of aromatic ring to one another will be accepted. The last line, tell to save the resulting tree, in a file called MyTree.tre.

14.5 T2C

The command line is:

```
T2C [options] -L <# of desired clusters>
```

Valid options are:

- -h [-help] : produces help message
- -N [-Nexus] arg : Nexus file name
- -O [-Outname] arg (=Out.clu) : Output cluster file/tree [Default=Out.clu]
- -L [-Level] arg (=1) : Level at which to cut the tree

The output contains on each line, the name of a leaf of the input tree and an integer referencing the cluster it belongs to.

Chapter 15

Interpretations of the output fingerprints

The fingerprints are based on the recognition of some interactions. Each atom bears a certain number of flags describing which kind of potential interactions it can be involved in. An interaction is detected if two atoms bear compatible flags and have a compatible geometry. This section describes how each flag is assigned and what are the expected geometries.

15.1 Flag and geometric descriptor assignment

The table 15.1 describes the different flags that can be assigned:

Note that the definition of an hydrogen bond and a weak hydrogen bond is not very well established. We have kept our definition as simple as possible for hydrogen bonds.

15.2 Geometry

Between a couple of atoms, an interaction is detected if the corresponding flag are compatible and if some geometric criteria (table 15.2) are respected between their geometric descriptors.

Flag	SMARTS
Donor	[O,N,S][H]
Acceptor	[O,N,*-;!+]
Cation	[*+]
Anion	[*-]
Aromatic	[a;R]
Hydrophobe	[C,S,F,Cl,Br,I]
Weak Acceptor	[a:a,A=A,A#A,S+0]
Weak Donor	[c,CX3,CX2][H]
Metal	[Mg,Ca,Mn,Fe,Co,Ni,Cu,Zn,Cd]

Table 15.1: For each flag defined in the long fingerprint the rule for assignment is given using Daylight SMARTS.

Interaction	Rule 1	Rule 2
Hydrogen bond	$ \overrightarrow{HA} \leq 3.5\text{\AA}$	$\widehat{\overrightarrow{DH}, \overrightarrow{HA}} \in [-\frac{\pi}{4}, +\frac{\pi}{4}]$
Weak Hydrogen bond	$ \overrightarrow{HA} \leq 2.8\text{\AA}$	$\widehat{\overrightarrow{DH}, \overrightarrow{HA}} \in [-\frac{\pi}{6}, +\frac{\pi}{6}]$
Ionic	$ \overrightarrow{+-} \leq 4.0\text{\AA}$	
Hydrophobe	$ \overrightarrow{Y_1Y_2} \leq 3.0\text{\AA}$	
Metal	$ \overrightarrow{M_1M_2} \leq 2.8\text{\AA}$	
Face to Face aromatic	$ \overrightarrow{ac_1ac_2} \leq 4.0\text{\AA}$	$\widehat{\overrightarrow{n_1}, \overrightarrow{n_2}} \in [-\frac{\pi}{6}, \frac{\pi}{6}]$
Edge to Face aromatic	$ \overrightarrow{ac_1ac_2} \leq 4.0\text{\AA}$	$\widehat{\overrightarrow{n_1}, \overrightarrow{n_2}} \in [\frac{\pi}{6}, \frac{5\pi}{6}]$
Pi-Cation	$ \overrightarrow{ac+} \leq 4.0\text{\AA}$	$\widehat{\overrightarrow{n}, \overrightarrow{ac+}} \in [-\frac{\pi}{6}, \frac{\pi}{6}]$

Table 15.2: Geometric conditions for an interaction to be detected. H stands for hydrogen, A , for acceptor, D for donor, $+$ for cation, $-$ for anion, Y for hydrophobe, M for a metal, ac for geometric center of an aromatic ring, n for the normal to the aromatic ring.

Position in the bit vector	Cavity atom Flag	Ligand atom flag	Interaction
1	Hydrophobe	Hydrophobe	Hydrophobe
2	Aromatic	Aromatic	Face to face
3	Aromatic	Aromatic	Edge to face
4	Donor	Acceptor	Hydrogen bond
5	Acceptor	Donor	Hydrogen bond
6	Cation	Anion	Ionic
7	Anion	Cation	Ionic
8	Weak donor	Weak acceptor	Weak hydrogen bond
8	Weak donor	acceptor	Weak hydrogen bond
9	Weak acceptor	Weak donor	Weak hydrogen bond
9	acceptor	Weak donor	Weak hydrogen bond
10	Aromatic	Cation	cation
10	Cation	Aromatic	cation
11	Metal	Metal	Metal

Table 15.3: Bit significance by residue

15.3 Fingerprint structure

The fingerprint is composed of 7 or 11 bits long blocks, one per residue in the cavity. The structure of each of those blocks is described in the table 15.3.

Note: manual setting of an interaction range to 0 Å will result in the suppression of the corresponding bits in the output.

Chapter 16

Conclusion

This manual presents a library and a few applications that derived from them. Those libraries are far from perfect and might need to be heavily rewritten for better conceptualization or optimization. But I believe that along with this manual, it becomes easy to manipulate, understand and maintain.

Some more work still waits. For example, several interaction centers should be possible, and more interaction descriptions should be examined.

Chapter 17

Support

For any question regarding installation, modification and use of the present libraries and applications, please contact:

Dr. G. Marcou (gilles.marcou@pharma.u-strasbg.fr)

Dr. D. Rognan (didier.rognan@pharma.u-strasbg.fr)

Input and output files are presented for exemplification in the **example** directory.

The programs and libraries can be downloaded at:

<http://bioinfo-pharma.u-strasbg.fr>

It is in the Download menu.