

CCCPart User's guide

Version v1.2 - Nov. 2006

1	Presentation	2
2	High level shell scripts	2
2.1	Preparing genome CDS files : C3P_EmbIToFasta.....	2
2.2	Computing gene homologs : C3P_OrthoBlast	4
2.3	Particular case of a single genome : C3P_OrthoSingle.....	6
2.4	Computing syntons : C3P_Syntons.....	6
2.5	Computing metabolons : C3P_Metabolons	8
2.6	Computing interactons : C3P_Interactons	11
3	Intermediate level shell scripts	14
3.1	Computing multigraphs from external files : C3P_MultiGraphxx.....	14
3.1.1	C3P_MultiGraphOrtho	14
3.1.2	C3P_MultiGraphKegg	14
3.1.3	C3P_MultiGraphPPI	15
3.2	Composing multigraphs : C3P_MultiGraphCompose	15
3.3	Computing the CCC partition : C3P_Partition	15
3.4	Examples of usage	15
3.4.1	C3P_Syntons	16
3.4.2	C3P_Metabolons.....	16
4	Low level binaries	16
4.1	Dimacs-like graph format.....	16
4.2	Computing multigraph partition : ccc_part.....	18
4.3	Computing partial transitive closure : ccc_closure.....	19
5	Annex 1 - Installation procedure	20

Contacts :

Frederic.Boyer@inrialpes.fr
Alain.Viari@inrialpes.fr

1 Presentation

The CCCPart (C3P) package implements a generic approach to merge the information from two or more graphs representing biological data, such as genomes, metabolic pathways or protein-protein interactions, in order to infer functional coupling between them. The theory is described in (1) and will not be detailed here (we strongly suggest you to read the paper before). The purpose of this documentation is to provide information about the package implementation and practical usage.

(1) : Syntons, Metabolons and Interactons: an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data.

F. Boyer, A. Morgat, L. Labarre, J. Pothier and A. Viari
Bioinformatics **21**:4209-4215 (2005)

The C3P is composed of three levels of shell scripts and executables.

Lower level contains C sources and binaries implementing the basic CCC (Common Connected Components) multigraph partitioning algorithm as described in (1). The input and output of these binaries are fairly general (Dimacs graph format) and can therefore be used for other purposes than comparative genomics.

Intermediate level is composed of Shell scripts acting as drivers of the lower level binaries and targeted at comparative genomic tasks. These scripts are mostly useful to bioinformaticians who want to devise their own computational strategies.

Higher level is composed of Shell scripts implementing "ready to use" strategies as described in (1). These scripts are intended for bioanalysts who want to use/test the package as quick as possible without programming.

For pedagogical reasons, these three levels will be now described in the reverse order (i.e. from higher level to lower level).

In the following examples, we will assume that the package has already been properly installed and fully tested (see the INSTALL file at the distribution root for more information) and that the C3P scripts are available in your path.

e.g. for csh :

```
$ set path = ($path <DISTRIBUTION_ROOT>/scripts) ; rehash
```

Since C3P may virtually produce a large number of result files, we recommend to create a temporary directory and to copy the samples files in there before proceeding.

```
$ mkdir c3ptemp
$ cd c3ptemp
$ cp <DISTRIBUTION_ROOT>/samples/* .
```

2 High level shell scripts

2.1 Preparing genome CDS files : C3P_EmbIToFasta

Since the main purpose of C3P is to deal with genomic data, the first kind of information to provide is the gene in the chromosomes under study. The CDSs of complete chromosomes should be provided in **fasta** format and formatted in the following way :

>CDSName CDSRank [Comments]
CDSTranslation

CDSName : any string (without blank or tab) identifying the CDS
Note: the CDSName's should be unique

CDSRank : an integer indicating the rank of the CDS on the chromosome (counting or not other non protein genes)

Note: the CDSrank starts at 1 and should be unique

Note: the CDSranks do not need to be contiguous (there might be holes in the numbering)

Note: the CDSs do not need to be sorted by ranks in the fasta file

Comments : optional comments on the CDS. For some applications (metabolons and interactons) this comment may be structured in the following way :

EC:<number>.<number>.<number>.<number> : is interpreted as an EC Number

SP:<string-identifier> : is interpreted as a Swiss-Prot accession number

CDSTranslation : is the amino-acid translation of the CDS

Note: the translation may run on several lines.

example:

```
>b0001.thrL 1 SP:P03059 Thr operon leader peptide
MKRISTTITTTITITTGNGAG
>b0002.thrA 2 EC:1.1.1.3 EC:2.7.2.4 SP:P00561 aspartokinase/homoserine
dehydrogenase I
MRVLKFGGTSVANAERFLRVADILESNNARQQGVATVLSAPAKITN
TLWKLG
>b0003 4 unknown protein
MVKVYAPASSANMSVGFVDVLGAAVTPVDGALLGDVVTVEAAETFS
WLGKNYLQNQEGFVHICRLDTAGARVLEN
```

In order to help you creating these files, we provide the **C3P_Emb1ToFasta** shell script that convert an EMBL formatted file to this fasta file format. By default, it produces CDSNames of the form <locus>.<geneName> and no comment.

```
--> reformat an EMBL file to fasta format suitable for C3P.

usage: C3P_Emb1ToFasta {emblFile}+ [-v verboseLevel] [-ec] [-sp] [-go]
      verboseLevel : 0 = no verbosity
                    1 = warning messages (default)
                    2 = all debug messages
      -ec : add EC numbers in comments
      -sp : add Swiss-Prot references in comments
      -go : add GO terms in comments
```

If you want the EC and SP information to be collected as well you should use the **-ec** and **-sp** options. However these options only work with EMBL GenomeReviews formatted files.

GenomeReviews files are available for download at :

<http://www.ebi.ac.uk/GenomeReviews/files>

Important Note: Unlike standard Unix utilities, C3P scripts take the options after the regular arguments.

Note: all C3P scripts accept a **-h** argument that provides help on script usage.

sample session:

all files mentionned here are provided in the samples directory of distribution and should be copied first to the current working directory as explained before.

```
$ C3P_Emb1ToFasta buaph.dat mygen.dat mypne.dat escol.dat -ec -sp
```

Don't worry about warning messages about pseudogenes (they don't have translation of course).

This will produce 3 new fasta formatted files named :

mygen.fst mypne.fst and escol.fst

Note that, as with the other C3P scripts, you have no control over the output file names, they are automatically defined by the scripts, from the input file names.

2.2 Computing gene homologs : C3P_OrthoBlast

The first step in C3P analysis is to establish an homology (ideally orthology) relation between the CDSs of two or more genomes. We provide a very simple strategy to do this by using sequence similarity and the popular BlastP approach. This is of course an approximation and other approaches may be devised to this purpose (BBH, BH, COGs). The basis is just to compare sequences and to say that genes are homologuous if they are sufficiently similar. the 'sufficiency' is defined by sequence similarity constraints.

```
---> compute homologs between n (n>=2) genomes using BlastP.

usage: C3P_OrthoBlast fastaFile fastaFile+ [-H hitmax] [-c covmin]
      [-p pmax] [-i idmin] [-m matrix]
      hitmax : max number of BlastP hits (default = 3)
      covmin  : min value of hit coverage = (Lali/min(L1,L2)*100)
                (default = 80)
      pmax    : max BlastP P-value (default = 1e-30)
      idmin   : min % identity (default = 40)
      matrix  : BlastP matrix (default = BLOSUM62)
```

the various options are associated to Blast results filtering i.e. to the stringency of the sequence similarity relation :

pmax is the maximum P-value allowed

idmin is the minimum percentage of sequence identity

covmin is the minimum alignment coverage, calculated as :
(size-alignment / min(sizeQuery, sizeTarget) * 100.

Important note: you need Blast (NCBI v2) to be properly installed in order to use this script (more precisely the two executables 'blastall' and 'formatdb' should be available in your path).

sample session:

```
$ C3P_OrthoBlast mypne.fst mygen.fst
```

this will produce the two following files :

mygen.mypne.blastp.3.BLOSUM62.out : raw output of blastP.

'3' is the default max number of hits and 'BLOSUM62' is the default matrix used.

Note that the input filename have been sorted alphabetically before processing. This allow to get the same result and same output filename whatever the order of input filenames.

This raw output is kept in the current directory in order to avoid the time consuming Blast recomputation if you want to change the other (filtering) parameters. Actually if you rerun the script with -c, -p or -i parameters, the raw output file will be recovered and only the filtering step will proceed.

mygen.mypne.ortho : an **orthoFile** i.e a file establishing the CDS to CDS homology correspondance.

Note: because BlastP P-values depends upon the size of the blasted DB, the results may slightly differ upon the order of the blasted genomes. This is usually not critical but, in some cases, it may be important to know in which order the blastP are performed. If the genome are A, B (in lexicographic order) then the computation performed is : A(Query) vs B(DB).

example of orthoFile (for 2 genomes):

```
#
# Homologs from blastP
# hitmax=3 covmin=80 pmax=1e-30 idmin=40 matrix=BLOSUM62
#
## N 523 729
MG001.dnaN MPN001.dnaN 1 1 | EC:2.7.7.7 SP:P47247 | EC:2.7.7.7 SP:Q50313
MG002.NONE MPN002.NONE 2 2 | SP:P47248 | SP:Q50312
MG003.gyrB MPN003.gyrB 3 3 | EC:5.99.1.3 SP:P47249 | EC:5.99.1.3 SP:P22447
MG003.gyrB MPN122.parE 3 129 | EC:5.99.1.3 SP:P47249 | EC:5.99.1.- SP:P78016
MG004.gyrA MPN004.gyrA 4 4 | EC:5.99.1.3 SP:P47250 | EC:5.99.1.3 SP:P22446
MG004.gyrA MPN123.parC 4 130 | EC:5.99.1.3 SP:P47250 | EC:5.99.1.- SP:P75352
MG005.serS MPN005.serS 5 5 | EC:6.1.1.11 SP:P47251 | EC:6.1.1.11 SP:P75107
MG006.tmk MPN006.tmk 6 6 | EC:2.7.4.9 SP:P47252 | EC:2.7.4.9 SP:P75106
MG007.NONE MPN007.NONE 7 7 | SP:P47253 | SP:P75105
```

'#' lines are comments

'##' line is mandatory. it is structured in the following way :

'N' maxRank1 maxRank2

maxRank1 : is the maximum rank in genome 1

maxRank2 : is the maximum rank in genome 2

other lines read as :

CDSName1 CDSName2 rank1 rank2 ['|'] [comments for 1] ['|'] [comments for 2]]

Note: the comments are optional

Important note: the CDS-CDS relationship is not one-to-one. i.e. one gene can have several homologs

Important note: More than 2 genomes can be provided to C3P_OrthoBlast. In that case, all two by two comparisons will be performed first and C3P_OrthoBlast will then assemble the results as n-tuples (where n is the number of genomes) by making **cliques** of orthologs. In other terms all pairs of CDS's in a n-tuple should satisfy the sequence similarity constraint.

sample session:

```
$ C3P_OrthoBlast mypne.fst mygen.fst buaph.fst
```

the orthoFile is now named : **buaph.mygen.mypne.ortho** (note the lexicographic order)

example of orthoFile for 3 genomes:

(optional comments have been omitted for clarity)

```
#
# Homologs from blastP
# hitmax=3 covmin=80 pmax=1e-30 idmin=40 matrix=BLOSUM62
#
## N 599 523 729
BU001.gidA MG379.gidA MPN557.gidA 1 432 597
BU006.atpA MG401.atpA MPN600.atpA 6 456 641
BU008.atpD MG399.atpD MPN598.atpD 8 454 639
```

2.3 Particular case of a single genome : C3P_OrthoSingle

The orthoFile plays a central role in C3P. In some case, however we may want to work with a single genome (e.g. for metabolons or interactons). For symmetry, we should also build an orthoFile in that case (the name is weird since there is no orthology here).

```
---> reformat a single genome fasta file to ortho file.
```

```
usage: C3P_OrthoSingle {fastaFile}+
```

sample session:

```
$ C3P_OrthoSingle escol.fst
```

this will produce an orthoFile named : **escol.ortho**

example of orthoFile for 1 genome:

```
#
# Genes from escol.fst
#
## N 4286
b0001.thrL 1 | SP:P03059
b0002.thrA 2 | EC:1.1.1.3 EC:2.7.2.4 SP:P00561
b0003.thrB 3 | EC:2.7.1.39 SP:P00547
b0004.thrC 4 | EC:4.2.3.1 SP:P00934
```

2.4 Computing syntons : C3P_Syntons

Now we have orthoFiles at hand, we can tackle the first task of C3P : computing syntons between $n \geq 2$ genomes. This will be illustrated here in the case of bupah, mygen and mypne.

```
---> compute syntons for n (>= 2) genomes

usage: C3P_Syntons orthoFile [-d delta] [-c | -l]
      delta : delta on genomic graph (default = 0)
      -c    : circular chromosome(s)
      -l    : linear chromosome(s) (default = -c)
      -o    : keep gene order
```

the **delta** parameter allows for 'gaps' in the primary graphs (see (1))

the **-l** option indicate that the genomes are linear. by default they are considered to as circular in order to compute the genoic interval graph (i.e. last ranks will wrap to first ranks).

the **-o** option (new in version 1.1) will force gene order conservation (i.e. no permutation allowed with $\text{delta} = 0$). In this case the delta parameter should be interpreted in a slightly different way : it indicates both the maximal number of 'gaps' and the maximal distance between two permuted genes.

Note: in this version there is no way to indicate that only one genome is linear. But you may implement this by using intermediate level scripts.

sample session:

```
$ C3P_Syntons buaph.mygen.mypne.ortho -d 5
```

this will produce two files :

buaph.mygen.mypne.ortho.5.graph : the multigraph file (not so useful for now)

buaph.mygen.mypne.ortho.5.part : the CCC partition (syntons) file

syntons partition file:

```
XX
XX Multigraph of homologs from : buaph.mygen.mypne.ortho
XX n=119 m=368 colors=3
XX
XX some size statistics...
XX
XX Classes sorted by min[elementSize] :
XX
CL score=15 nodeSize=15 min[eltSize]=15 eltSize : 15 15 15
VE BU499.rpoA MG177.rpoA MPN191.rpoA 491 187 198 | | EC:2.7.7.6 | EC:2.7.7.6
VE BU501.rpsK MG176.rpsK MPN190.rpsK 493 186 197 | | SP:P47422 | SP:Q50296
VE BU502.rpsM MG175.rpsM MPN189.rpsM 494 185 196 | | SP:P47421 | SP:Q50297
VE BU503.rpmJ MG174.rpmJ MPN188.rpmJ 495 184 195 | | SP:P47420 | SP:P52864
VE BU504.secY MG170.secY MPN184.secY 496 180 191 | | SP:P47416 | SP:Q59548
VE BU507.rpsE MG168.rpsE MPN182.rpsE 499 178 189 | | SP:P47414 | SP:Q50301
VE BU510.rpsH MG165.rpsH MPN179.rpsH 502 175 186 | | SP:P47411 | SP:Q50304
VE BU512.rplE MG163.rplE MPN177.rplE 504 173 184 | | SP:P47409 | SP:Q50306
VE BU514.rplN MG161.rplN MPN175.rplN 506 171 182 | | SP:P47407 | SP:Q50308
VE BU517.rplP MG158.rplP MPN172.rplP 509 168 179 | | SP:P47404 | SP:P41204
VE BU518.rpsC MG157.rpsC MPN171.rpsC 510 167 178 | | SP:P47403 | SP:P41205
VE BU519.rplV MG156.rplV MPN170.rplV 511 166 177 | | SP:P47402 | SP:P75575
VE BU520.rpsS MG155.rpsS MPN169.rpsS 512 165 176 | | SP:P47401 | SP:P75576
VE BU521.rplB MG154.rplB MPN168.rplB 513 164 175 | | SP:P47400 | SP:P75577
VE BU524.rplC MG151.rplC MPN165.rplC 516 161 172 | | SP:P47397 | SP:P75580
```

```

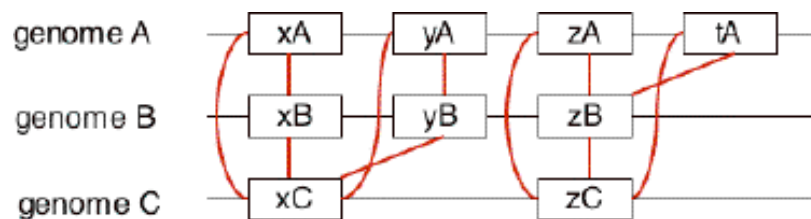
XX
CL score=3 nodeSize=3 min[eltSize]=3 eltSize : 3 3 3
VE BU527.fusA MG089.fusA MPN227.fusA 519 94 234 | | SP:P47335 | SP:P75544
VE BU528.rpsG MG088.rpsG MPN226.rpsG 520 93 233 | | SP:P47334 | SP:P75545
VE BU529.rpsL MG087.rpsL MPN225.rpsL 521 92 232 | | SP:P47333 | SP:P75546
XX

```

Each **CL** line introduce a class (i.e. a synton) and following **VE** lines detail the nodes. Each VE line correspond to a n-tuple of orthologous genes with names, ranks and comments.

The syntons are sorted by numbers of genes. Here, the largest synton is composed of 15 genes (ribosomal proteins). However, since the same gene can appear more than once in a synton (if it is related to more than one gene in another genome), we choose to measure the size of synton as the minimum number of different genes in each genome. This is best explain using an example.

The following synton (red lines indicates similarity relationship) :



has a **score** of 2 (2 different genes in genome C) and will print as :

```

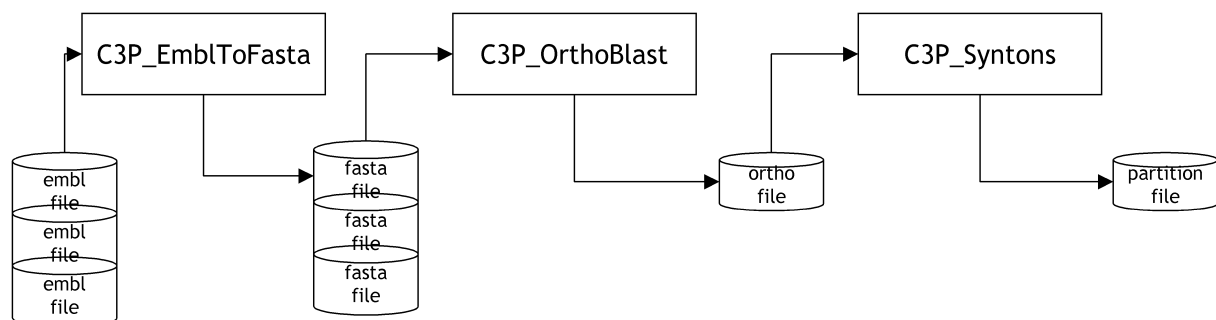
CL score=2 nodeSize=4 min[eltSize]=2 eltSize : 3 4 2
VE xA xB xC rxA rxB rxC
VE yA yB xC rxA rxB rxC
VE zA zB zC rxA rxB rxC
VE tA zB zC rxA rxB rxC

```

The **nodeSize** is the actual size of the CCC (nodes of the multigraph) and the eltSize give the number of different genes on each genome.

Summary:

The following schema summarizes the Syntons calculation process



2.5 Computing metabolons : C3P_Metabolons

We can now turn to the second task of C3P : computing metabolons. To do this, we need 3 things : i) an orthoFile (either from C3P_OrthoBlast or C3P_OrthoSingle) ii) a metabolic network and iii) a correspondence between genes and metabolic reactions.

The **C3P_metabolons** script implements a strategy where the metabolic network is extracted from the KEGG 'reaction' file and the correspondance is given by EC numbers.

```

---> compute metabolons

usage: C3P_Metabolons orthoFileWithEC KeggReactionFile [-d deltaG] [-D
deltaR] [-C cutoff] [-c | -l]
    deltaG : delta on genomic graph (default = 0)
    deltaI : delta on interaction graph (default = 0)
    cutoff : compound connectivity cutoff (default = 20)
    -c      : circular chromosome(s)
    -l      : linear chromosome(s) (default = -c)

```

the **orthoFile** should contain 'EC:' in comments. i.e it should have been constructed with fasta sequences containing EC numbers in comments (see 2.1).

the **KeggReactionFile** can be downloaded from the KEGG site : \$to be completed\$. It has the following format :

```

ENTRY      R00001
NAME        Polyphosphate polyphosphohydrolase
DEFINITION  Polyphosphate + H2O <=> Oligophosphate
EQUATION    C00890 + C00001 <=> C02174
RPAIR       A02844
ENZYME      3.6.1.10
///

```

Note: only the ENTRY, EQUATION, ENZYME and // lines are mandatory.

the **deltaG** and **deltaR** parameters are the delta parameters on the genomic and reaction graph respectively.

the **cutoff** parameter is used to exclude compounds that are connecting too many reactions (ubiquitous compounds). default is 20.

sample session:

```
$ C3P_Metabolons escol.ortho reaction.kegg -d 1
```

this will produce 4 files :

```

reaction.kegg.20.graph           : the reaction graph
escol.ortho.1.graph              : the genomic graph (delta = 1)
escol.ortho.1.reaction.kegg.20.0.graph : the correspondance multigraph
escol.ortho.1.reaction.kegg.20.0.part : the CCC partition (metabolons)

```

metabolons partition file (1 genome):

```

XX
XX Multigraph composed :
XX
XX Multigraph of homologs from : escol.ortho
XX
XX Multigraph reaction from : reaction.kegg
XX n=1995 m=92047 colors=2
XX
XX size statistics

```

```

XX
XX Classes sorted by min[elementSize] (ignoring columns 2) :
XX
CL score=8 nodeSize=17 min[eltSize]=8 eltSize : 8 (17)
VE b0092.ddlB R01150 94 | EC:6.3.2.4 SP:P07862 | 6.3.2.4
VE b0091.murC R03193 93 | EC:6.3.2.8 SP:P17952 | 6.3.2.8
VE b0090.murG R05662 92 | EC:2.4.1.227 SP:P17443 | 2.4.1.227
VE b0090.murG R05032 92 | EC:2.4.1.227 SP:P17443 | 2.4.1.227
VE b0090.murG R05027 92 | EC:2.4.1.227 SP:P17443 | 2.4.1.227
VE b0088.murD R02783 90 | EC:6.3.2.9 SP:P14900 | 6.3.2.9
VE b0087.mraY R05630 89 | EC:2.7.8.13 SP:P0A6W3 | 2.7.8.13
VE b0087.mraY R05629 89 | EC:2.7.8.13 SP:P0A6W3 | 2.7.8.13
VE b0087.mraY R05628 89 | EC:2.7.8.13 SP:P0A6W3 | 2.7.8.13
VE b0086.murF R04617 88 | EC:6.3.2.10 SP:P11880 | 6.3.2.10
VE b0086.murF R04573 88 | EC:6.3.2.10 SP:P11880 | 6.3.2.10
VE b0086.murF R04572 88 | EC:6.3.2.10 SP:P11880 | 6.3.2.10
VE b0085.murE R02788 87 | EC:6.3.2.13 SP:P22188 | 6.3.2.13
VE b0084.ftsI R06179 86 | EC:2.4.1.129 SP:P04286 | 2.4.1.129
VE b0084.ftsI R06178 86 | EC:2.4.1.129 SP:P04286 | 2.4.1.129
VE b0084.ftsI R06177 86 | EC:2.4.1.129 SP:P04286 | 2.4.1.129
VE b0084.ftsI R04519 86 | EC:2.4.1.129 SP:P04286 | 2.4.1.129
XX
CL score=6 nodeSize=25 min[eltSize]=6 eltSize : 6 (19)
...
XX

```

Again the classes (metabolons) are scored and sorted by min[eltSize] but in this case column 2 (i.e. the reaction) is ignored. This means that metabolons are scored by number of different genes involved. In this example, the first (largest) metabolon is composed of 17 nodes involving 8 different genes and 17 different reactions.

The previous remark is important because of the degeneracy of the EC numbers. Some EC are indeed related to a lot of reactions (some of them possibly not appearing in the organism under study). So a better way would be to related directly the genes to the reactions they catalyses. Unfortunately this information is not present in the current databases.

Finally, one can also compute metabolons by using several genomes by simply supplying the corresponding orthoFile. However since the homologs in each n-tuple may have different annotated EC number, one should define more precisely how the mapping is done. In the C3P_Metabolons strategy, we choosed to perform an OR between EC's of homologs in each genome. i.e a reaction should match at least one gene's EC.

sample session:

```
$ C3P_Metabolons mygen.mypne.ortho reaction.kegg
```

will produce the partition file : **mygen.mypne.ortho.0.reaction.kegg.20.0.part**

metabolons partition file (2 genomes):

comments have been omitted for clarity

```

XX
XX Multigraph composed :
XX
XX Multigraph of homologs from : mygen.mypne.ortho
XX
XX Multigraph reaction from : reaction.kegg

```

```

XX n=548 m=9016 colors=3
XX
XX some statistics...
XX
XX Classes sorted by min[elementSize] (ignoring columns 3) :
XX
CL score=4 nodeSize=16 min[eltSize]=4 eltSize : 4 4 (5)
VE MG271.pdhD MPN390.pdhD R01698 301 411 |
VE MG271.pdhD MPN390.pdhD R01698 301 411 |
VE MG272.pdhC MPN391.pdhC R02569 302 412 |
VE MG272.pdhC MPN391.pdhC R02569 302 412 |
VE MG273.pdhB MPN392.pdhB R00014 303 413 |
VE MG273.pdhB MPN392.pdhB R00014 303 413 |
VE MG273.pdhB MPN392.pdhB R01699 303 413 |
VE MG273.pdhB MPN392.pdhB R01699 303 413 |
VE MG273.pdhB MPN392.pdhB R03270 303 413 |
VE MG273.pdhB MPN392.pdhB R03270 303 413 |
VE MG274.pdhA MPN393.pdhA R00014 304 414 |
VE MG274.pdhA MPN393.pdhA R00014 304 414 |
VE MG274.pdhA MPN393.pdhA R01699 304 414 |
VE MG274.pdhA MPN393.pdhA R01699 304 414 |
VE MG274.pdhA MPN393.pdhA R03270 304 414 |
VE MG274.pdhA MPN393.pdhA R03270 304 414 |
XX
CL score=2 nodeSize=6 min[eltSize]=2 eltSize : 2 2 (3)

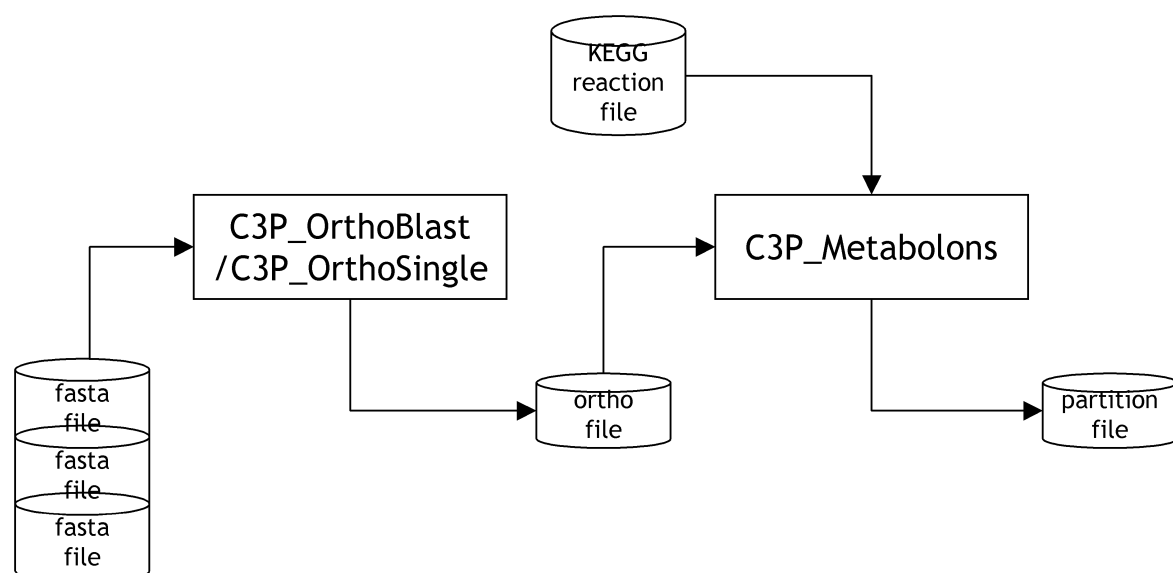
```

Note that column 3 (reaction) is, again, omitted in the statistics and score.

In this case the largest metabolon is composed of 16 nodes corresponding to 4 different genes in mypgen, 4 different genes in mypne and 5 different reactions.

Summary:

The following schema summarizes the Metabolons calculation process



2.6 Computing interactons : C3P_Interactons

The third task is to compute interactions. To do this, we need 3 things : i) an orthoFile (usually obtained by using C3P_OrthoSingle), ii) a protein-protein (PPI) interaction network and a correspondence between gene in the orthoFile and protein names in the PPI network.

the C3P_Interactions script implements a strategy where the PPI network is extracted from DIP tabulated files and the correspondence is given by Swiss-Prot accession numbers.

```

---> compute interactions

usage: C3P_Interactions orthoFileWithSP DIPTabulatedFile
        [-d deltaG] [-D deltaI] [-k keyword] [-c | -l]
    deltaG : delta on genomic graph (default = 0)
    deltaI : delta on interaction graph (default = 0)
    keyword : keyword to retrieve SP entry in DIP file (default = "SWP")
    -c      : circular chromosome(s)
    -l      : linear chromosome(s) (default = -c)

```

the **orthoFile** should contain 'SP:' in comments. i.e it should have been constructed with fasta sequences containing SP: Swiss-Prot accession numbers in comments (see 2.1).

the **DIPTabulatedFile** can be downloaded from the DIP site : \$to be completed\$. It has a tabulated format where each line corresponds to one interaction. The actual content of the line does not matter as long as it contains the two following mandatory fields :

'SWP:'<swiss-prot-acnum1> 'SWP:'<swiss-prot-acnum2>

only these two fields are used to retrieve the accession numbers of interacting proteins.

sample DIP tabulated file :

```

DIP:58E DIP:339N SWP:P06139 PIR:BVECGI GI:7429025 DIP:1081N SWP:P04273 PIR:UJHYIH GI:2144854
DIP:61E DIP:1025N SWP:P00968 PIR:SYECCP GI:68663 DIP:1026N SWP:P00907 PIR:SYECCS GI:68268
DIP:62E DIP:539N SWP:P07622 PIR:MMECMG GI:72516 DIP:508N SWP:P02928 PIR:JGECM GI:72544
DIP:169E DIP:124N SWP:P21891 PIR:A39202 GI:96549 DIP:726N PIR:S30701 GI:421191
DIP:208E DIP:367N SWP:P07002 PIR:DEECXB GI:2144347 DIP:366N SWP:P07001 PIR:DEECXA GI:16129561
DIP:263E DIP:342N SWP:P13458 PIR:BVEESC GI:73012 DIP:572N SWP:P03702 PIR:QGBPL GI:76015
DIP:269E DIP:572N SWP:P03702 PIR:QGBPL GI:76015 DIP:540N SWP:P08394 PIR:NCECX5 GI:67271
DIP:270E DIP:583N SWP:P11191 PIR:RCBP22 GI:75879 DIP:540N SWP:P08394 PIR:NCECX5 GI:67271

```

the **deltaG** and **deltaI** parameters are the delta parameters on the genomic and PPI graph respectively.

sample session:

```

$ C3P_Interactions escol.ortho dip.tab -d 1

```

this will produce 4 files :

dip.tab.graph	: the PPI graph
escol.ortho.1.graph	: the genomic graph (delta = 1)
escol.ortho.1.dip.tab.0.graph	: the correspondance multigraph
escol.ortho.1.dip.tab.0.part	: the CCC partition (interactons)

interactons partition file:

```

XX
XX Multigraph composed :
XX
XX Multigraph of homologs from : escol.ortho

```

```

XX
XX Multigraph PPI from : dip.tab
XX n=342 m=246 colors=2
XX
XX some statistics...
XX
XX Classes sorted by min[elementSize] (ignoring columns 2) :
XX
CL score=7 nodeSize=7 min[eltSize]=7 eltSize : 7 (7)
VE b3738.atpB SWP:P00855 3656 | EC:3.6.3.14 SP:P00855 276
VE b3736.atpF SWP:P00859 3654 | EC:3.6.3.14 SP:P00859 275
VE b3735.atpH SWP:P00831 3653 | EC:3.6.3.14 SP:P00831 274
VE b3734.atpA SWP:P00822 3652 | EC:3.6.3.14 SP:P00822 273
VE b3733.atpG SWP:P00837 3651 | EC:3.6.3.14 SP:P00837 272
VE b3732.atpD SWP:P00824 3650 | EC:3.6.3.14 SP:P00824 271
VE b3731.atpC SWP:P00832 3649 | EC:3.6.3.14 SP:P00832 270
XX
CL score=6 nodeSize=6 min[eltSize]=6 eltSize : 6 (6)
VE b0742.ybgF SWP:P45955 731 | SP:P45955 66
VE b0741.pal SWP:P07176 730 | SP:P07176 65
VE b0740.tolB SWP:P19935 729 | SP:P19935 64
VE b0739.tolA SWP:P19934 728 | SP:P19934 63
VE b0738.tolR SWP:P05829 727 | SP:P05829 62
VE b0737.tolQ SWP:P05828 726 | SP:P05828 61
XX

```

As for Syntons and Metabolons, the classes (Interactons) are scored and sorted by min[eltSize]. As for Metabolons, the column 2 (i.e. the protein) is ignored. This is quite useless here since, usually, they should be as many different proteins as different genes. The only case where a different can occur is when two (adjacent) genes -encodes for (exactly) the same protein (i.e. with the same SP accnum).

In this example, the first (largest) interacton is composed of 7 nodes involving 7 different genes and 7 different proteins.

Finally, it is possible to run C3P_Interacton with a multiple genome orthoFile. This mau look a little bit strange since a PPI network is usually defined only for one species and, therefore, the gene-protein correspondence will onlu work for this species. The C3P_Interactons script will usually detect this. For instance if you try :

```
$ C3P_Interactons mygen.mypne.ortho dip.tab
```

You will get the following message :

```

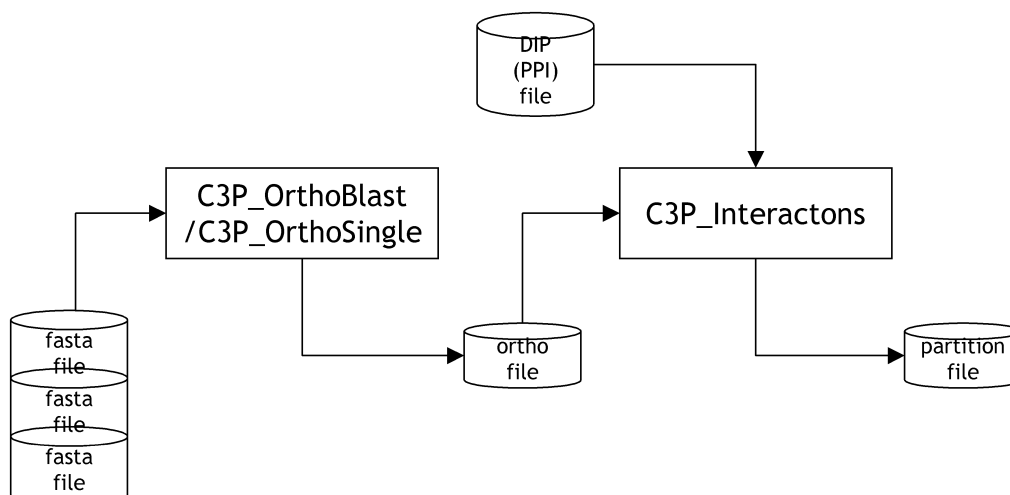
*
* none of the homologs's SP matches a PPI SP
* are you sure the ppi species is one of the orthofile species ?
*

```

However, if you want to experiment with this, you should just know that C3P_Interacton performs the same type of correspondence as C3P_Metabolons, that means an OR between the SP numbers of homologs in each genome. i.e a protein should match at least one gene's SP number.

Summary:

The following schema summmarizes the Interactons calculation process



3 Intermediate level shell scripts

These scripts are mostly used for designing alternate strategies. They are intended for Unix programmers and their usage is documented in the script source.

3.1 Computing multigraphs from external files : C3P_MultiGraphxx

There are currently 3 shell scripts to compute a multigraph from external source files : orthofile, Kegg reaction file and DIP interaction file.

3.1.1 C3P_MultiGraphOrtho

```

---> compute multigraph from orthofile with n (>= 2) genomes

usage: C3P_MultiGraphOrtho orthoFile [-d delta] [-c | -l]
      delta : delta on genomic graph (default = 0)
      -c    : circular chromosome(s)
      -l    : linear chromosome(s) (default = -c)

input: orthoFile
output: $orthoFile:r.$delta.graph : correspondence multigraph (Dimacs
      format)
  
```

3.1.2 C3P_MultiGraphKegg

```

---> compute a reaction graph from Kegg reaction file

usage: C3P_MultiGraphKegg KeggReactionFile [-C cutoff]
      cutoff : compound connectivity cutoff (default = 20)

input : (KeggReactionFile) : KEGG style like with 3 mandatory lines :
output: $KeggReactionFile.$cutoff.graph : reaction graph (Dimacs format)
  
```

3.1.3 C3P_MultiGraphPPI

```
---> compute an protein-protein interaction graph from DIP-like tabulated
file

usage: C3P_MultiGraphPPI DIPTabulatedFile [-k <keyword>]
      keyword : keyword to retrieve SP entry in DIP file (default = "SWP")

input: (DIPTabulatedFile) : DIP tabulated file (.tab) with one interaction
      per line and 2 mandatory fields marked with "SWP:" keyword (Swiss-
      Prot reference). (the default keyword ("SWP") can be changed using
      the -k option)

output: DIPTabulatedFile.graph : interaction graph (Dimacs format)
```

3.2 Composing multigraphs : C3P_MultiGraphCompose

This script will compute the composition of 2 multigraphs to make a new multigraph. The script takes the two input multigraph as input and a file that lists the elements of the restriction of the cartesian product of their nodes (in other terms, the correspondence between their nodes). The file format is simply two numbers per line, each for the node index in the corresponding multigraph.

```
usage: C3P_MultiGraphCompose multiGraphFile1 multiGraphFile2
      restrictionList

input: multiGraphFile1, multiGraphFile2 : multigraphs in Dimacs format
input: (restrictionList) : a list of couples indicating the restriction of
the cartesian product multiGraphFile1 x multiGraphFile2

output: $multiGraphFile1:r.$multiGraphFile2:r.graph : composed multiGraph
      (Dimacs format)
```

Note: the script will keep and join properly the multigraph node comments.

3.3 Computing the CCC partition : C3P_Partition

```
---> compute CCC partition from multigraph

usage: C3P_Partition multiGraphFile [-i tuplePos] [-elt | -ccc]
      tuplePos: tuple position to ignore in statistics. positions should
                be separated by '|' e.g. -i "1|4" (default = "")
      -elt    : sort classes by element size
      -ccc    : sort classes by CCC size (default -ccc)

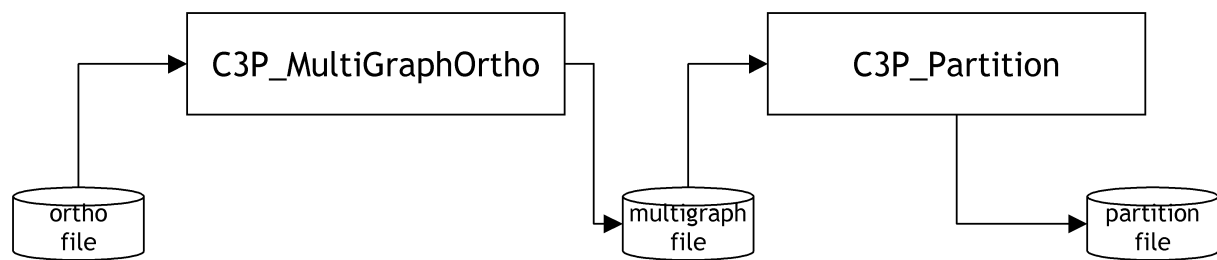
input: multiGraphFile (Dimacs format)
output: $multiGraphFile:r.part : pretty output of partition file
```

3.4 Examples of usage

We will illustrate the use of intermediate scripts on the example of Syntons and Metabolons.

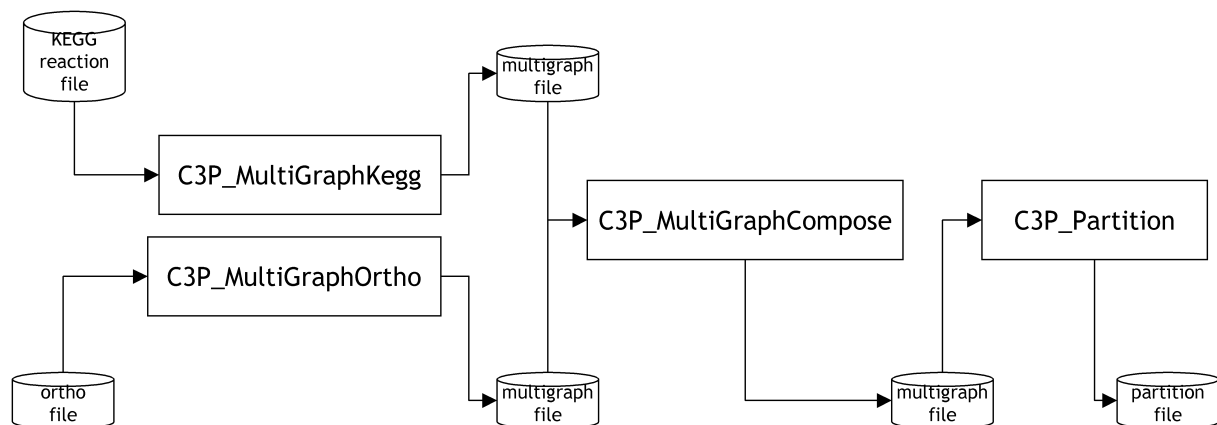
3.4.1 C3P_Syntons

C3P_Syntons can be computed in the following way :



3.4.2 C3P_Metabolons

C3P_Metabolons can be computed in the following way :



4 Low level binaries

The low level binaries are composed of two executables: **ccc_part** and **ccc_closure**. The input and output of these binaries are fairly general (Dimacs graph format) and can therefore be used for other purposes than comparative genomics.

4.1 Dimacs-like graph format

A Dimacs graph file is composed of 3 parts (that should appear in order) :

a single 'p' line // mandatory
a series of 'n' lines // optional
a series of 'e' lines // mandatory

Each line is composed of fields separated by any number of blanks or tab characters.

the 'p' line

the 'p' line contains exactly 6 fields :

```
'p' 'grf' <#Nodes> <#Edges> <#Colors>
```


'grf' : mandatory keyword

<#Nodes> : number of nodes

<#Edges> : number of edges

<#Colors> : number of different colors (for multigraph)

the 'n' lines

each 'n' line contains 3 or more fields :

'n' <nodeNum> <nodeLabel> [<nodeComments>]

<nodeNum> : a 1-based index for the node (<nodeNum> \in [1, <#Nodes>])

<nodeLabel> : a string (without blanks or tabs) indicating the node label

<nodeComments> : any number of strings indicating comments

Note: the <nodeNum> should be unique, in the range [1, <#Nodes>] and there should be exactly <#Nodes> 'n' lines.

Note: the <nodeLabel> does not need to be unique

Note: the 'n' lines are optional for using the ccc binaries
(they are however required in C3P scripts)

the 'e' lines

each 'e' line contains exactly 3 fields :

'e' <nodeNum1> <nodeNum2> <edgeColor>

<nodeNum1> : the index of node 1

<nodeNum2> : the index of node 2

<edgeColor> : binary representation of edge colors.

This is a string composed of '0' and '1' characters only and the positions of 1's indicate the colors present.

For instance if the 'p' line says the graph is defined with 2 colors (let's say red and black) then

1 (or 01) : means red

10 : means black

11 : means red and black

Note: the 0 (or 00) color is forbidden

(it is actually equivalent to 'no edges' !)

Important note: the graph handled by ccc binaries are supposed to be **undirected**. Therefore, there is no need to duplicate the edges (e.g. to state 'e 1 2' and 'e 2 1'). Doing so will, however, have no impact on the result of the algorithm (this will just slow down the process).

Important note: the binary color scheme allow you to minimize the number of declared edges but it is actually equivalent from the algorithm point of view to declare one edge with k colors or k edges with one color. For instance the following declaration are equivalent :

e 1 2 11

or

e 1 2 1

```
e 1 2 10
```

or

```
e 1 2 10  
e 2 1 1
```

Again, although the results will be the same, the difference will be in execution time (first form will run quicker).

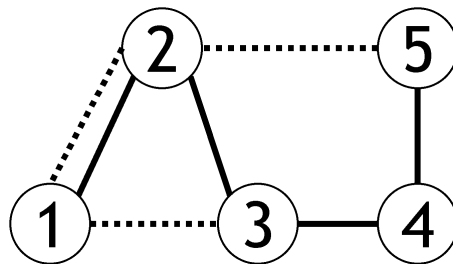
Note: you may add any number of comment lines by using the 'c' keyword :

```
'c' [<comments>]*
```

Example of Dimacs graph:

```
c  
c sample graph  
c  
p gpt 5 6 2  
c  
n 1 a  
n 2 b  
n 3 c  
n 4 d  
n 5 e  
c  
e 1 2 11  
e 1 3 01  
e 2 3 10  
e 2 5 01  
e 3 4 10  
e 4 5 10
```

this correspond to the following undirected graph :



4.2 Computing multigraph partition : ccc_part

ccc_part implements the CCC partitioning algorithm

```
ccc_part < inputGraph > outputPartition
```

input: a Dimacs graph

output : a partition file with the following format

```
'c' 'main niter='<numberOfIterations>
'c partition k='<numberOfClasses>
```

then a series of k classes (one per line) :

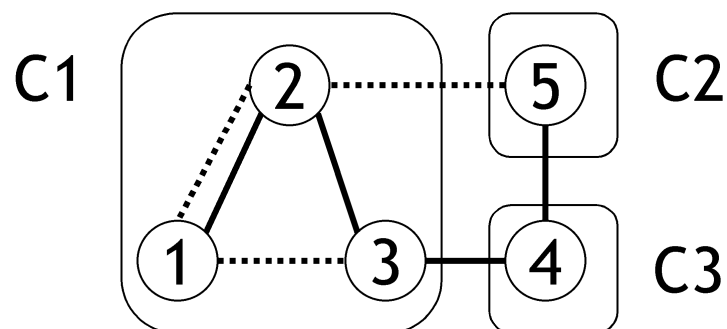
```
'q' <classeIndex> ':' <nodeNum>+
```

Example of partition file:

ccc_part with the previous graph will produce :

```
c main niter=2
c partition k=3
q 1 : 3 2 1
q 2 : 5
q 3 : 4
```

corresponding to the following partition :



4.3 Computing partial transitive closure : ccc_closure

```
ccc_closure -d delta < inputGraph > outputGraph
```

input: a Dimacs graph

output : a Dimacs graph - partial transitive closure of input

Important note : ccc_closure currently works only for *unicolor* graphs (i.e. with <#Colors> = 1). If you provide a multicolored multigraph, the program will ignore the edge colors and compute the partial closure on all edges (it issues a warning however).

5 Annex 1 - Installation procedure

1. C sources and binaries

=====

This CCCPart distribution contains sources and pre-compiled binaries for Linux and MacOSX platforms.

The binaries are located in CCCPart/ports/<portname>/bin

where portname is one of :

i386-linux	: compiled on RedHat 9.\$ - i386
ppc-darwin	: compiled on MacOSX PPC 10.4
i386-darwin	: compiled on MacOSX INTEL 10.4
mips-irix	: SGI Irix
sparc-solaris	: SPARC Solaris

Therefore on these machines, you don't need to recompile the binaries and you can jump to the next section.

If you need/want to [re]compile for one of these ports, just issue

make

from the distribution root directory (CCCPart).

Note: If you need to configure CCCPart for another port, please consult Annex 1

Note: to recompile, you need an ANSI C (e.g. gcc) compiler

2. Shell scripts and path

=====

Shell scripts (for csh) are located in CCCPart/scripts

You can call them directly in this location from any other location or you may add the 'CCCPart/scripts' directory in your path.

Note that you don't need to add the CCCPart/ports/<portname>/bin directory in your path qince the scripts will locate the proper binaries to use at runtime.

If you prefer to copy the scripts in another directory (e.g. /usr/local/bin), then

you *should* also copy the binaries at the same location. We recommend you to do so

only after checking that everything runs fine (see next section).

```
cp -r scripts/* /usr/local/bin          # you may need to be root to do so
cp ports/<portname>/bin/* /usr/local/bin # you may need to be root to do so
```

note: don't forget the '-r' option when copying scripts !

3. Testing the distribution

=====

We recommend to perform this step before using the software.

From the distribution root directory (CCCPart), just issue :

make test

The test takes approximately 5 minutes to run.

If everything runs fine, it should terminates with :

```

+
+ -----
+ All Tests were succesfull
+ -----
+

```

If this is the case, then everything is ok, so you can jump to next section.

The test requires Blast (NCBI v2) to be properly installed (in order to establish sequence similarity in C3P_OrthoBlast). If this is not the case, the test will first issue :

```

* -----
* Blast failure at : C3P_OrthoBlast
* -----
* The most likely reason is that NCBI Blast is not properly installed
* You should have the 'blastall' and 'formatdb' executables in your path
* -----
*

```

the test will continue (after you type <return>) but will eventually terminate with :

```

+
+ -----
+ 1 Failure(s) encountered
+ -----
+

```

If you are sure that blastall is properly install, this error may simply arise from slight numerical differences in BlastP outputs on different platforms. In this case, you can ignore the warning.

4. clean the distribution

=====

After tests run fine, you may want to cleanup the unnecessary files.

From the distribution root directory (CCCPart), just issue :

```
make clean
```

5. using the software

=====

A pdf User's guide is located in CCCPart/docs as :

C3PUsersGuide.pdf

Please read this one first.

```

-----
In case of trouble, you may send an email at :
Alain.Viari@inrialpes.fr
-----

```

=====

Annex 1

=====

Here are the steps to install the software on a new port. Since this has not be done before (!), the following instructions may need some adaptation.

-a- cd to 'config' directory

-b- issue './guess_port' to get your port name
if the answer is 'unknown-port' then you need to edit the 'guess_port' script to add a new portname (try to use the same convention : <CPU>-<SYSTEM>)

- c- if the <portname>.conf file does not exist, then you should create one. the best way is to start from the closest existing .conf file, to copy it and edit it to fit with your system.
- d- go back to distribution root directory (..) and issue 'make'

when everything's compiled, try the binaries by running :

./tests/CTests/allTest.csh
- e- if test runs fine, then copy the 'config/guess_port' script (in case you have modified it) to 'scripts/C3PUtils/C3PGuessPort'