Project Tracking System for Automated Structure Solution Software Pipelines

Wanjuan Yang, Ronan Keegan and Peter Briggs

CCP4, CSE Department, STFC Daresbury Laboratory, Warrington WA4 4AD

1. Introduction

In recent years, there has been an increasing number of projects focusing on the development of automated structure determination software "pipelines" covering various aspects of the post-data collection stages of structure solution by protein X-ray crystallography (PX). Examples of these include MrBUMP[1], Autorickshaw[2], Crank[3], HAPPy[4], XIA2[5].

Generally pipelines work in a similar way to how an experienced scientist would work – they make choices about which programs to run and try to select optimal parameters by analysis of the input data and the outputs of the underlying software. The automated systems are therefore of help to both less experienced users, who may get better results than by using the same programs manually, and for expert users where the automated procedure can perform routine tasks quickly and free them to spend more time on problems that requires their expertise.

As automated systems run it is typical that they can often perform many discrete steps and generate large amounts of data, which must be accurately recorded and tracked to ensure successful operation of the pipeline. However it presents a challenge to the end user who needs to be able to interpret the final results and understand how they were obtained.

In this article we present an overview of a project tracking system which is intended to help with the management and presentation of the data within automated pipelines. We describe the dbCCP4i system which provides generalised tools to store information about the individual steps or program runs performed within automated PX structure solution software, and to present this data in an intuitive manner to the end user. The technical details of the system are described along with a set of code fragments to demonstrate how it can be incorporated into existing Python or Tcl scripts, and two case studies are presented describing practical usage of the system, including incorporation of the tracking system into the current release of MrBUMP.

2. Overview of the dbCCP4i Project Tracking System

The architecture and components that comprise the dbCCP4i project tracking system has already been described in some detail in a previous CCP4 newsletter article [6], however it is still useful to give a brief overview. The key components are shown below in figure 1 and are described in the following sections.

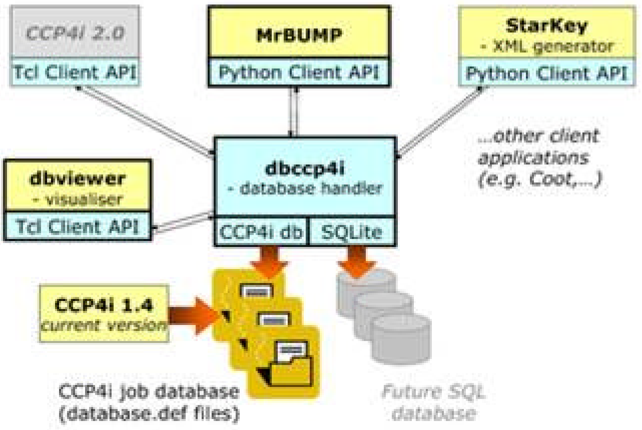


Figure 1: Architecture of project tracking system Project database handler

2.1 Project database handler

At the core of the system is the project database handler dbCCP4i, which is essentially a small server program that handles interactions between the database and other programs.

It processes requests from "client applications" (that is, an automated pipeline or any other program that wishes to store or retrieve data from the database) and interacts with the database on their behalf by performing storage and retrieval actions in response to their requests.

Using the handler hides much of the details of the database (described in the next section) from the client programs. It also allows multiple applications to access a single database simultaneously. The clients can be informed of changes to the database made by other programs via "broadcast messages" sent to all clients by the handler.

The handler is written in Python programming language, however the communications have been kept as language-neutral as possible by using a simple pseudo-XML markup to describe requests and responses, and by passing these between the handler and the clients via sockets. In practice, since the technical details of dealing with the socket communications can become quite complicated, "client API" libraries are provided to hide these details and simplify access to the database. Use of the client API libraries is described later on in section 3.

The system runs on both UNIX/Linux and MS Windows platforms.

2.2 Database for storing job data

The dbCCP4i system is built on the existing database used within the graphical user interface CCP4i [7], This records basic information about each program run (also called a "job"), including the date, input parameters, associated input and output files, and the outcome of the run.

The data itself is stored using the same "database.def" file infrastructure as CCP4i, and so is backwardly compatible with existing data already stored by users of CCP4i. This means that for example the visualisation tool described in the following sections can be used to look at old and new projects run by the user.

We now briefly describe how job record data is recorded within CCP4i, for those not already familiar with the system.

Within CCP4i users can organise their work into "projects". A project is simply a collection of data files plus an associated job database. Conventionally the data within a particular project will be associated with the same structure solution or part of a structure solution, however users or application programs are free to define projects in any way that they wish – there is no formally enforced definition of a project.

Within CCP4i and dbCCP4i, each of these projects is identified by a unique name or "alias" that is used as an identifier elsewhere in the system. The projects are implemented as a directory which contains the files in the project plus the database.def file which holds the raw job tracking data. The "job database" for each project contains job records, where a job is a run a software application or task (which itself can be a single program or a script that runs many programs in some sequence).

Each job record has the following information:

- An automatically assigned job id number
- A last modification date andtime
- A "task name" corresponding to the name of the program, task or script
- An associated status indicating success or failure
- An arbitrary user – or application-specified title
- The name of any assocated files, including: a log file, run file, and lists of input and output files

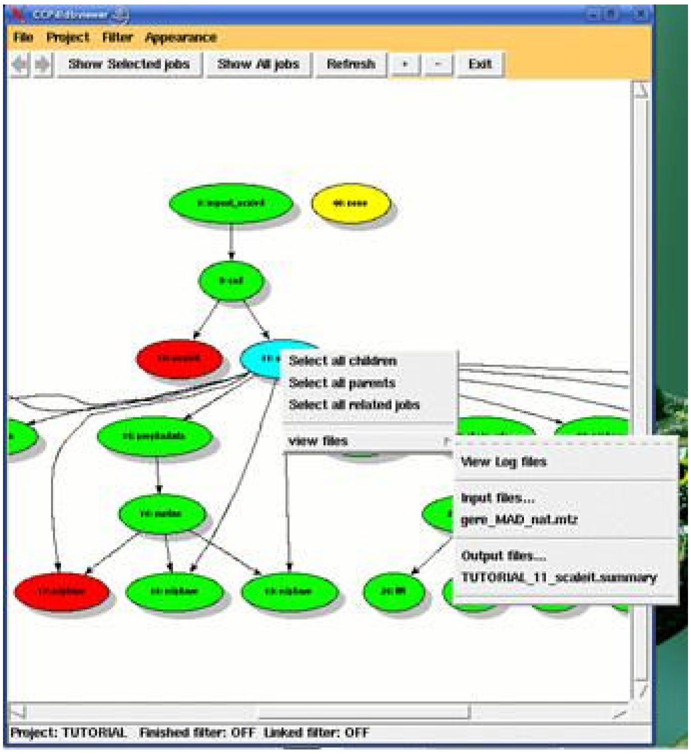
All these data items can be accessed via functions provided in the client APIs.

2.3 Visualisation of the job data using dbviewer

Users already familiar with CCP4i will be aware that the job database can be examined and interacted with via the chronological "job list" that is presented as part of CCP4i’s main interface window. While this list-based view is useful, it also has some limitations and it can quickly become difficult to navigate the project history when a large number of jobs have been run, and also doesn’t give any indication of how the jobs might be related to each other.

As part of the dbCCP4i system an alternative way of visualising the project history has been developed within the dbviewer program. Dbviewer presents the tracking data in an intuitive manner to the end user by showing the jobs within the project with links between them indicating the transfer of data. The user can interact with the view to see the files and other data associated with each job (see the figure 2).

The viewer is written in Tcl/Tk (for compatibility with CCP4i) and uses the Graphviz package [8] as an engine for generating the layouts of the history graphs that can be seen in the figure.



Project: TUTORIAL Finished filter: OFF Linked filter: OFF

Figure 2: dbviewer: jobs with the links indicating the transfer of data files

Within dbviewer each job is represented by a "node" in the graph. Jobs are linked together by lines wherever an output file from one job is used as input to another job, to show the flow of data through the project. The colour of each node provides an indication of the status of the job, specifically green for completed jobs, red for failed jobs, and yellow for those which are still running.

The viewer also provides tools to interact with the job data as described in the following sections.

2.3.1 Selection tools

Dbviewer provides tools for users to select a subset of jobs to view, so that users can concentrate on the jobs that are of particular interest to them. For example, the visualiser can trace all the "antecedents" or "descendents" of a particular job selected by the user. Alternatively the user can manually select a subset of jobs to view. The user can also step back or forward through previous selections.

2.3.2 Filtering tools

The filtering tool allows users to change the view so that the unsuccessful or unconnected jobs are filtered out of the view.

2.3.3 Viewing the associated files

By right clicking on a job in the viewer, users can access a menu of associated files that are available to view, specifically the log file and the input and output files for the selected job.

2.3.4 Generate xml file for structure deposition

Using the selection tool, users are able to select the jobs which lead to the successful structure solution. Dbviewer then offers an option to write out the job information in an XML file which could be used for structure deposition. The XML file is generated using starKey program, which is discussed in a later section as an example of a client application which accesses the databases.

2.3.5 Real-time monitoring of progress of automated pipelines

Where the project tracking system has been incorporated into an automated program in order to store the jobs or steps that have been run within the pipeline, dbviewer can act as a monitor to watch the progress of the automated progress. As steps complete and their status changes, or new steps are added, dbviewer updates its display accordingly.

The case study integrating dbCCP4i the MrBUMP automated MR system is described in a later section and allows the dbviewer to be used in this way.

3. Using the project tracking system in PX software via the Client API libraries

Within dbCCP4i a set of "client API" libraries are provided which have been developed to make it easier for application programs to interact with the database. So far client APIs have been developed in Tcl and in Python. This section describes how the client API functions can be used, with the aid of example scripts in both Python and Tcl.

Before a client application program can access the database it must establish a connection to the database handler (example fragment #1 for Python programs, and #6 for Tcl scripts). When establishing the connection, the application can supply its name (shown as the text "ApplicationName" in these examples) – this is not currently used, but in future will be stored as part of any new records or other data added to the database by the application.

Once the connection has been established, the application can use it to access the data in the database – acquiring a list of the existing project names, opening an existing project, or create a new project (each shown in example fragments #2 and #7). (Note that it is possible for an application to have multiple projects open at any one time.) In the examples, the project name "myproject" is used.

After the application has opened a particular project (note that creating a new project automatically opens it), it can perform various operations on the job data in that project, as shown in examples #3 and #4 for Python and #8 and #9 for Tcl.

Examples of adding a new job to an open project is shown in examples #3 and #8. Jobs are referenced by unique integer id numbers within each project, so in each case the result of the operations shown should be a new id number for that project. This id number is then used when retrieving or modifying the data associated with that job – for example in #4 and #5, there are examples of setting the job title and status, and associating a program logfile and other input and output files.

Data can also be retrieved from the database. The examples in #4 and #9 show how to get a list of the jobs already defined in the database, how to retrieve job data such as title, status, and associating input and output files.

Finally, when the application has finished its interactions with the database, it should ideally close all the open projects and disconnect from the database handler, as shown in example fragments #5 and #10 for Python and Tcl respectively. In practice the system is robust enough that the handler can cope with unexpected disconnections by client applications (for example if the client crashes), still it is recommended to disconnect cleanly.

Python script:

1. How to initialise the connection to the handler

To talk to dbhandler, The following codes need to be included in the applications:

```
# import the client API
import dbClientAPI

# create a HandlerConnection instance, connect to the handler
conn = dbClientAPI.HandlerConnection(‘ApplicationName’,True)
```

2. How to list the available projects & open/create a project

After connecting to the handler, you are ready to access the information in the database. For example, you want to know what the projects are available, do the following:

```
# list available projects
conn.ListProjects()
```

If you want to open an existing project, you need to give the project name.

```
# open an existing project
conn.OpenProject("myproject")
```

If you want to create a new project, give the project name and directory that the project is going to be hold.

```
# create a new project
conn.CreateProject("myproject","/home/project/myproject")
```

3. How to add a job

After you open an existing project or create a new project, you could add new jobs data or update the old jobs data. Adding a new job will return a jobid which you need to use for later updating or accessing the job data.

```
# add a new job
conn.NewJob("myproject")
```

4. How to modify and retrieve data for a job

The following codes show how to update or retrieve the job data.

```
# list all jobs in a project
conn.ListJobs("myproject")
```

```
# set job title, status etc
conn.SetTitle("myproject",jobid,"Refine mr coordinates")
conn.SetStatus("myproject",jobid,"FINISHED")
```

```
# add input/output files, logfile
conn.SetLogfile("myproject",jobid,"10_refmac5.log")
conn.AddInputFile("myproject",jobid,"abc.mtz")
conn.AddOutputFile("myproject",jobid,"molrep1_refmac1.pdb")
```

```
# retrieve input/output files
conn.ListInputFiles("myproject", jobid)
conn.ListOutputFiles("myproject", jobid)
```

```
# retrieve job title, status etc
conn.GetData("myproject",jobid,"TITLE")
conn.GetData("myproject",jobid,"STATUS")
```

5. How to finish up

After finishing accessing the database, you need to close all the projects that you opened and disconnect from the dbhandler.

```
# close project
conn.CloseProject("myproject")
```

```
# close connection
conn.HandlerDisconnect()
```

The following is the equivalent Tcl script:

Tcl script:

1. How to initialise the connection to the handler

```
# import the client API
source dbClientAPI.tcl
```

```
# Start the handler
DbStartHandler handlebroadcast
```

```
# Connect to the handler
DbHandlerConnect "ApplicationName" True
```

2. How to list the available projects & open/create a project

```
# list available projects
ListProjects
```

```
# open an existing project
OpenProject "myproject"
```

```
# create a new project
CreateProject "myproject" "/home/project/myproject" msg
```

3. How to add a job

```
# add a new job, return a job id
set jobid [NewJob "myproject" ]
```

4. How to modify and retrieve data for a job

```
# list all jobs in a project
ListJobs "myproject"

# set job title, status etc
SetData "myproject" jobid TITLE "Refine mr coordinates"
SetData "myproject" jobid STATUS "FINISHED"

# add input/output files, logfile
SetData "myproject" jobid LOGFILE "10_refmac5.log"
AddInputFile "myproject" jobid "abc.mtz"
AddOutputFile "myproject" jobid "molrep1_refmac1.pdb"

# retrieve input/output files
ListInputFiles "myproject" jobid
ListOutputFiles "myproject" jobid

# retrieve job title, status etc
GetData "myproject" jobid STATUS
GetData "myproject" jobid TITLE
```

5. How to finish up

```
# close project
CloseProject "myproject"

# close connection
DbHandlerDisconnect
```

4. Case Study

This section describes client applications that use dbCCP4i.

4.1 Case I: MrBUMP

MrBUMP is a framework that automates the determination of macromolecular structures via the Molecular Replacement method(MR). In MR, a trial model based on a known structure is used as a template for extracting the structural information of the target from the experimental reflection data. The success of the MR procedures is critically dependent upon the choice of trial model, and so the approach used in MrBUMP emphasises the generation of a variety of search models.

In good cases MrBUMP will give a single clear solution to an MR problem; in other cases it will suggest a number of likely search models that can be investigated manually by the scientists – typically around 10-15 search models, although it could be as much as ten times that. In preparation for molecular replacement each model structure also undergoes rounds of processing by various programs such as side-chain pruning based on sequence similarity between the target and template. The modifications are then carried out using various CCP4 programs such as Chainsaw or Molrep.

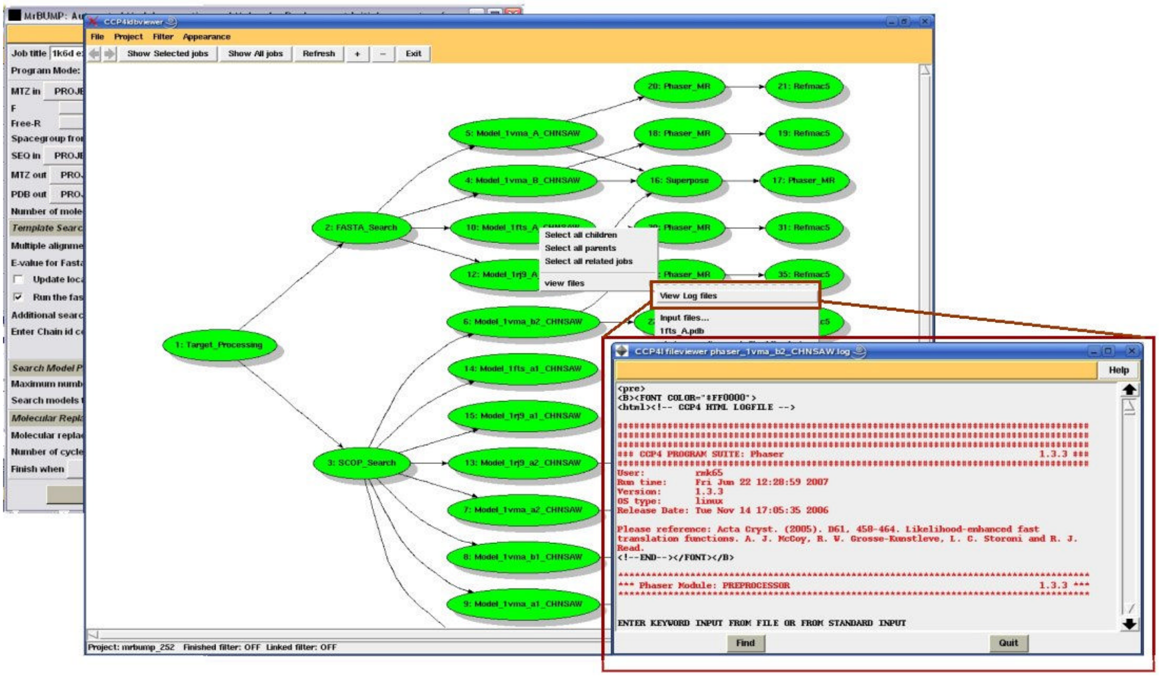
As a result the end user faces a challenge when reviewing and evaluating the results of the MrBUMP run, and in earlier releases end users needed to filter through many different output files and directories.

In order to address this problem, the most recent release of MrBUMP(0.4.0) uses dbCCP4i to record information from each of the steps involved in the MR process and the dbviewer is then used to present the data in a more intuitive graphical form. The graphical view makes it much easier for the end user to monitor the progress of the MrBUMP run, and to more easily find and examine a particular file associated with the processing of each search model. Thus the use of the dbCCP4i substantially improves the ease of use of MrBUMP for the end user.

It is important to note that MrBUMP uses the node-job representation in the graphical viewer in a slightly modified way: rather than representing specific CCP4 jobs, nodes in the viewer are used more loosely to represent stages in the course of the automatic pipeline whilst still providing access to the underlying log and data files. A typical run of MrBUMP breaks down into the following set of steps:

- Processing of the input target data.
- Carrying out the various searches (sequence, secondary structure, domain, multimer)
- Multiple alignment of target with search model sequences.
- Downloading search model PDB files.
- Preparing search models (side-chain prunings, superposition of structures, etc.)
- Molecular replacement.
- Refinement of molecular replacement results.
- Phase improvement.

Each of these steps can involve the use of several CCP4 or third-party programs and can occur multiple times depending on the number of search models used, so it makes more sense to represent the steps as nodes rather than the individual jobs. This helps to provide a more intuitive and informative view of what is happening in the pipeline. This is made possible by the large degree of flexibility provided by the dbCCP4i API.



```
<pre>
<B><FONT COLOR="#FF0000">
<html><!-- CCP4 HTML LOGFILE -->

###################################################################################
###################################################################################
###################################################################################
### CCP4 PROGRAM SUITE: Phaser                                           1.3.3 ###
###################################################################################
User:         rmk65
Run time:     Fri Jun 22 12:28:59 2007
Version:      1.3.3
OS type:      linux
Release Date: Tue Nov 14 17:05:35 2006

Please reference: Acta Cryst. (2005). D61, 458-464. Likelihood-enhanced fast
translation functions. A. J. McCoy, R. W. Grosse-Kunstleve, L. C. Storoni and R. J.
Read.
<!--END--></FONT></B>

***********************************************************************************
*** Phaser Module: PREPROCESSOR                                          1.3.3 ***
***********************************************************************************


ENTER KEYWORD INPUT FROM FILE OR FROM STANDARD INPUT
```

Figure 3: Example of the dbviewer showing the output from a MrBUMP job

4.2 Case II: starKey

Deposition of the structure with the wwPDB is an important part of the process of structure determination. Since dbCCP4i in principle allows diverse structure determination programs to access the same database, it should at some stage be possible for users to perform structure determination using a mixture of manual CCP4i tasks and automated pipeline programs and graphical model building tools, and have all the data history appear in the same project. It would then be possible at the end of the structure determination to collect together all the relevant data based on the job history in order to feed this directly into deposition.

To this end, starKey is a small program that has been developed as part of the dbCCP4i project, and which is able to collect the data history information from selecting the job nodes that lead to the final solution, and then using the "Generate XML" option previously described. This runs the starKey program with a project name and a list of jobs, which starKey then uses to extract the relevant data from the database in order to populate its XML file. In principle the resulting XML file could one day be used for structure deposition.

4.3 Case III: dbviewer

Dbviewer is another example of client application of the db handler. It is written in Tcl/Tk and talks to db handler via Tcl client APIs which described in section 3. The detail of dbviewer is described in section 2.3.

5. Availability

Version 0.2 of dbCCP4i is available for download from ftp://ftp.ccp4.ac.uk/bioxhit/dbccp4i-0.2.tar.gz

It includes the dbviewer and starKey applications as well as the core dbCCP4i handler and client APIs.

The system requires you to have CCP4 suite version 6.0 or newer, plus python 2.4 or newer and Graphviz package is required for the dbviewer. The dbviewer is compatible with the current CCP4i database, and can be run stand-alone.

MrBUMP version 0.4.0 incorporating dbCCP4i is available for download from www.ccp4.ac.uk/MrBUMP

This also includes the viewer and handles the installation of the various dbCCP4i dependencies.

6. Conclusions and Future Directions

The project tracking system is available for general CCP4 usage and could easily be used by other automation projects. The use of project tracking system means the outcome of an automation program is much clearer and accessible to the users. In addition, the Dbviewer can be used as a monitor to view the real-time progress of an automation pipeline.

There still remain some issues with the cohesion of MrBUMP's usage of dbCCP4i and the CCP4i interface but these are currently being worked upon. In particular, the issue of how to represent a MrBUMP job within the framework of a CCP4i project needs to be resolved. To address this, a MrBUMP job or any other automation pipeline will be represented as a sub-job or sub-project within a CCP4i project. When viewing the contents of a CCP4i project, a MrBUMP job will be represented by a single node in the graph. Double-clicking it will spawn a secondary viewer, detailing the contents of the MrBUMP job.

The dbCCP4i system is built on existing job database used in CCP4i. The system is compatible with current version of CCP4i. Thus CCP4i users can use dbviewer to look at the existing CCP4i projects as well as new projects created in CCP4i.

While the initial version of tracking database provides storage for job data which is useful for applications, one possible future direction is to combine this job history data with crystallographic data to produce a so-called "rich database". In particular, to provide a knowledge base which consists of the common crystallographic data items that are used in a software pipeline and can be shared between different applications. Currently we are working on a small version of the knowledge base with the intention of demonstrating the usage of the knowledge base in applications. The demonstration version of the knowledge base consists of dataset and heavy atom sub structure information which is used in the CCP4i "MLPHARE" task. An interface for storing and retrieving the data will be provided. We are also working on integrating dbCCP4i and dbviewer into CCP4i.

Finally, we are very keen to work with any automation software developers who would like to integrate dbCCP4i into their automation programs. Any feedback is very welcome.

7. Acknowledgements

This work is on dbCCP4i has largely been undertaken within the BIOXHIT project, which is funded by the European Commission within its FP6 Programme, under the thematic area "Life sciences, genomics and biotechnology for health", contract number LHSG-CT-2003-503420. Both the dbCCP4i and MrBUMP project have also been generously supported by STFC Daresbury Laboratory via the CCP4 project.

WY and PJB would also like to acknowledge the various contributors who have helped by providing many useful suggestions via discussions, emails and testing - in particular Ronan Keegan, Martyn Winn, Charles Ballard, Graeme Winter, Daniel Rolfe and Steven Ness.

8. References

[1] R.M.Keegan and M.D.Winn, *Acta Cryst*. **D63**, 447-457 (2007), "Automated search-model discovery and preparation for structure solution by molecular replacement"

http://www.ccp4.ac.uk/MrBUMP/

[2] S.Panjikar et al, *Acta Cryst*. **D61**, 449-457 (2005), "Auto-Rickshaw - An automated crystal structure determination platform as an efficient tool for the validation of an X-ray diffraction experiment"

http://www.embl-hamburg.de/Auto-Rickshaw/

[3] S.R.Ness et al, *Structure* **12**. 1753-1761 (2004), "Crank: New methods for automated macromolecular crystal structure solution"

http://www.bfsc.leidenuniv.nl/software/crank/

[4] HAPPy: P.Emsley, D.R.Rolfe and C.C.Ballard (unpublished) http://www.ccp4.ac.uk/HAPPy/

[5] G.Winter, *BSR 2007* poster, "Application of Expert Systems to MX Data Reduction";

CCP4 Newsletter 2007, "XIA2 – A brief user guide", (in this Issue)

http://www.ccp4.ac.uk/xia/

[6] P.J.Briggs and W.Yang, *CCP4 Newsletter* 45 (2007), "CCP4 in BIOXHIT: Managing and Visualising Project Data"

http://www.ccp4.ac.uk/newsletters/newsletter45/articles/ccp4_bioxhit.html

[7] E.Potterton et al, *Acta Cryst*. **D59**, 1131-1137 (2003), "A graphical user interface to the CCP4 program suite"

http://www.ccp4.ac.uk/ccp4i_main.php

[8] Graphviz: "Graph Visualization Software"

http://www.graphviz.org