M. Sc. Thesis

Semi-automatic level scheme solver for nuclear spectroscopy

Kaj Jansson

Department of Physics
Division of Nuclear Physics
Lund University
February 2011
Acknowledgements

To complete this thesis I have received help, from a number of people, who deserve to be acknowledged. First I would like to thank my supervisor, Joakim Cederkäll, for discussions as well as continuous help and support.

A special thanks goes out to the helpful people at the RootTalk support forum who quickly answered most my questions regarding the ROOT library.

Finally I would also like to thank Jenny for proof reading the thesis and putting up with me during busy times.
Abstract

A software suite consisting of a program for peak finding and fitting called FifFi and a program for building level schemes called WhaC have been developed. To each program an eligible graphical interface was written. Using these programs several methods of background subtraction in $\gamma$-ray coincidence spectra were evaluated, among those a newly developed method, Median cuts on sub ranges, which showed good results on the tested spectra. The scheme builder was equipped with two different algorithms: a statistical one based on the Metropolis method and a logical one developed from scratch. These two methods are compared both on an ideal case and on experimental $\gamma$-ray coincidence data. The logical algorithm, heavily based on coincidences, is shown to be able to correctly solve schemes providing the supplied experimental coincidence data is correct and complete. Also, for incomplete data and data with minor errors the algorithm produces consistent sub schemes when no complete scheme is possible.
Contents

1 Introduction .................................................. 1

2 Background ....................................................... 3
   2.1 Nuclear structure ........................................... 3
   2.2 Nuclear $\gamma$-decay ....................................... 3
      2.2.1 Rotational bands ...................................... 4
   2.3 $\gamma\gamma$-coincidence experiments ........................ 5
      2.3.1 Fusion evaporation .................................... 5
      2.3.2 Coulomb excitation ................................... 5
      2.3.3 Detecting $\gamma$-rays ................................. 6
      2.3.4 Coincidence matrices ................................ 6

3 Program details ............................................... 9
   3.1 The ROOT library ............................................ 9
      3.1.1 C++ interpreter ....................................... 10
      3.1.2 Signals ................................................. 10
   3.2 Interface and features ..................................... 10
      3.2.1 Peak finder and fitter ................................ 11
      3.2.2 Level scheme builder .................................. 12
      3.2.3 I/O ..................................................... 12
   3.3 Class structure ............................................. 14
   3.4 Known bugs ................................................ 15

4 Finding peaks ................................................. 17
   4.1 Background subtraction .................................... 17
      4.1.1 Deconvolution ......................................... 17
      4.1.2 Peak clipping .......................................... 18
      4.1.3 Median cut on sub ranges ............................. 18
      4.1.4 Pixel exterminator .................................... 18
   4.2 Peak search algorithm ..................................... 19

5 Building a level scheme ..................................... 21
   5.1 Available information ..................................... 21
   5.2 Previous level scheme solvers ............................. 21
   5.3 Statistical approach ....................................... 22
      5.3.1 Metropolis .............................................. 23
      5.3.2 Simulated annealing ................................... 24
List of Figures

3.1 The program FifFi showing its tools for viewing the matrix as well as fitting peaks. .................................................. 11
3.2 The program WhaC after solving the scheme of a loaded set of transitions. . 13
3.3 A simplified class diagram of the program suite developed. Only the main classes, main methods and main relations are shown. ..................... 13
5.1 Flowchart symbol key ...................................................... 26
5.2 The Simulated annealing algorithm ........................................ 27
5.3 Construction of sub schemes in the buildScheme sub routine. The flowchart continues in fig. 5.6 .................................................. 28
5.4 The 100, 55 and 45 keV transitions form a ‘group-permutable’ that together with the 30 keV transition (a regular ‘permutable’) form a permutation group. The 55 and 45 keV transitions form a smaller permutation group contained within the larger one. .................................................. 30
5.5 All transitions except the 100 keV transition are part of the ‘flippable’. Due to the 25 keV transition in the middle no regular permutations are possible. . 30
5.6 Checking for scheme overlaps in the buildScheme sub routine. Continuation of the flowchart in fig. 5.3. The flowchart continues in fig. 5.6 and the merge sub routine can be seen in fig. 5.8 .................................................. 31
5.7 Handling parallel schemes in the buildScheme sub routine. Continuation of the flowchart in fig. 5.6 .................................................. 32
5.8 Merging sub schemes in the buildScheme sub routine. A sidetrack to the flowchart in fig. 5.6 .................................................. 33
5.9 Flowchart of the createMerge sub routine ................................. 34
5.10 Flowchart of the conditionalLevelMerge sub routine .................. 36
6.1 The cleaning algorithm removes the most background and leaves a clean histogram to the right. The original histogram can be seen to the left. . 40
6.2 Lower part of the $^{170}\text{Er}$ nucleus scheme solved by the scheme solver using perfect coincidence data .................................................. 41
6.3 The resulting scheme from running the metropolis routine on experimental coincidence data .................................................. 42
6.4 10 000 moves were performed for each data point. Starting on 1 000 the temperature was decreased by 1% after each set of moves until it reached 0.1. . 42
6.5 Scheme solver result of experimental dataset with missing information. . 43
6.6 The lower levels of $^{181}\text{Ta}$ found in the $^{170}\text{Er}$ experiment ................................. 44
Chapter 1

Introduction

Since the dawn of computers, science has been one of their most heavy users as an aid and tool in the search for more and better knowledge. Early computers did tedious and boring calculations much faster and more precise than humans. Today the computers still do this but computers and computer science have evolved to fill much more advanced purposes. But still they are just machines doing calculations.

Humans have specialized in high mental capacity through evolution and can still beat computers at certain tasks well suited for the human brain. Pattern recognition allows humans to solve complex problems without doing all the calculations a computer would do and can therefore sometimes be faster even though modern computers do billions of calculations each second. Chess is beaten, computers defeat humans by brute force and compete on the grandmaster level. The old Chinese game Go is still not beaten since the $19 \times 19$ possible positions and the more strategical, rather than tactical, play required are too much for today’s computers and they fall short before professional and experienced amateur human players [1].

In nuclear physics the matter of finding a level scheme based on a set of known transitions and their relations is a puzzle humans solve utilizing various tactics. Calculations, pattern recognition and intuition are all part of it but it takes experience and patience. So why not let someone, or something, do the hard work? Creating a level scheme of a few hundred transitions might seem like a lot but it is nowhere near the complexity of Go and the experimental data contains a lot more information to be used finding the level scheme structure.

There is also the interesting question whether humans, when solving a level scheme, use their experience too much. Humans tend to repeat previous patterns and might overlook other possibilities. The computer has no experience but is objective and does only what the program tells it to. Not only should a computer be able to come closer to an objective truth, but it also lacks the annoying habit of making the human errors the scientist has to quarrel with.

This thesis explores how the calculation power of today’s computers can be used to solve these level schemes so that humans can do more interesting stuff. Building a level scheme might not be considered boring the first few times but after a while it usually becomes repetitive and monotonous.

The first part of this thesis concerns finding peaks in large two, or more, dimensional matrices. The computer can do this better, more objective and faster than humans but it
will never understand what it is doing and might sometimes make errors only humans can recognize. A semi-automatic procedure allows the computer to do the boring tasks and the user may control or correct the results for particular complicated and odd exceptions.

The second part investigates the possibility to let the computer find the level scheme on its own. Also here the automatic procedure transcends into semi-automatic when something unpredicted happens, for example small parts of information can be missing. The users can then use their experience to try to correct the results or to lead the program onto the right path again. Any way it goes, the goal is to alleviate the scientist from tasks that is unnecessarily done by hand.

In the following chapters the peak search and level scheme building will be discussed along with the method tried and developed. The two programs developed forms a suite to handle experimental coincidence data but should be looked at as software for reviewing the developed methods rather than a finished product like the commonly used γ-ray analysis package Radware\(^1\).

The full codebase developed currently consist of approximately 11,000 lines of code which would fill hundreds of pages and is therefore not attached to this thesis. Readers interested in the implementation wanting to see the code are encouraged to contact the author.

\(^1\)http://radware.phy.ornl.gov
Chapter 2

Background

2.1 Nuclear structure

The levels in a nucleus are quantum mechanical states which can be populated or not by specific reactions. The state of the nucleus is the result of what quantum states all the protons and neutrons in the nucleus occupy. Each nucleon state will have an associated parity and angular momentum and as a consequence the total state of the nucleus will have the appropriate sum of all the nucleons’ properties. If a nucleon gets excited to a state of higher energy the whole nucleus also increases its energy. Some effects are best explained not as single nucleon properties but rather as collective states involving all the nucleons at once. What is observed in such a case as a nuclear level is a combination of single particle states and a collective state.

An important aspect of nuclear structure is the so called ‘magic numbers’ which denote the number of protons or neutrons that one can put into a nucleus before one reaches a large energy gap. That is, the nucleon coming right after a ‘magic number’ sits at a significantly higher energy level than the previous one. Understanding the ‘magic numbers’ means understanding which states that are favoured and what nuclei that can be produced, artificially or not, but requires studies of the experimentally observable nuclear structures. Learning about energies, angular momenta and parity of the nuclear levels can lead to better understanding of, for example, the strong force, cosmic events, fission and fusion.

Going between different states in a nucleus requires either energy to be released or absorbed. A nuclear decay goes from a higher state to a lower and releases energy simply by emitting it in form of an energy quantum with a specific momentum. As in all quantum mechanics the probability for a certain reaction depends on the overlap of the wave function, resulting from operating on the initial wave function with an appropriate operator, and the final wave function. One can therefore expect that angular momentum and parity will often have large impact on the decay modes.

2.2 Nuclear $\gamma$-decay

$\gamma$-decays are decay modes that simply send out the excess energy as a photon. The energy of the $\gamma$-ray is usually somewhere in the range tenths of keV to a few thousand keV. The radiation emitted can be of different multipole orders and of either electric or magnetic character. Generally the electric radiation is stronger than the magnetic but low multipole order
CHAPTER 2. BACKGROUND

is most important. These results come from the Weisskopf estimates [2] and give us a tool for determining which radiation to expect from which transition in the nucleus.

The parity of an electric dipole can easily be seen to be $-1$ since the transformation $x \rightarrow -x$ will switch the field direction. The same arguments determine the electrical quadrupole radiation to be of even parity since the transformation $x \rightarrow -x$ in this case does nothing. Examining the magnetic multipole radiation in the same way one comes to the conclusion that its parity is the opposite of the electric radiation of the same multipole order. This is an important result since parities give rise to a strong selection rule; if the nuclear states have different parities they can only be connected by radiation of odd parity, and vice versa.

The angular momentum conservation gives rise to another selection rule. As usual the angular momenta are added quantum mechanically so if the initial angular momentum of the nucleus is $I_i$ and that of the final state is $I_f$ the radiation must have a multipole order of $|I_i - I_f|, \ldots, I_i + I_f$. Depending on the values of $I_i$ and $I_f$ this can severely diminish the decay rate. Usually all decays are possible for at least one multipole order either electric or magnetic except in the case $I_i = I_f = 0$ when the transition has to occur utilizing some other mechanism than $\gamma$-decay.

2.2.1 Rotational bands

One should note that rotation of a nucleus is only possible if the nucleus is deformed in any way. Why? Because quantum mechanically a rotation around any symmetry axis would not really change the object. It would only result in a phase factor. Even a deformed nucleus can only rotate around an axis that is not a symmetry axis.

The rotation is a collective feature of the nucleus, not necessarily all nucleons must participate but it is done by a larger number of nucleons. The Hamiltonian of a rotating body is the angular momentum divided by two times the moment of inertia $H = \frac{J^2}{2I}$. If the rotation is completely collective i.e. all nucleons participate in the rotation, the angular momentum is the same as the the total angular momentum $I$ of the nucleus. Thus the energy of those rotational states is proportional to $I(I+1)$ in a normal quantum mechanical manner. The spacing between these levels will then be $2(I+1)$, assuming $J$ is constant and $\Delta I = 1$, leading to a predictable pattern of transition energies.

In some cases like nuclei with even numbers of protons and neutrons the wave function is invariant to a rotation by $\pi$. Such a rotation’s effect is to change the sign of the angular momentum projection, $\Omega$, on the symmetry axis. To make the wave function invariant during a rotation of $\pi$ it will have to be a superposition of the states having $\pm \Omega$. If the total angular momentum projection on the symmetry axis is zero, as in even-even nuclei, this puts restrictions on $I$ since the wave function can be shown to be proportional to $1 + (-1)^I$ [3]. Thus sometimes the symmetry only allows $I$ to increase in steps of two resulting in the energy gap being proportional to $2(2I+3)$ instead.

More complicated descriptions of rotation involves the coupling between the angular momenta of the nucleons positioned outside a shell, where the angular momenta are coupled to zero, and the collective rotation. The angular momenta of the nucleons will be forced to align themselves to the rotational axis as the rotation gets stronger and a similar relationship between the resulting transitions are found. These bands, with their regular changes in the transition energies, provide great help to physicists interpreting experimental results.

The decay down towards the ground state from a high excited state often goes by these bands. The state with the lowest energy for a given spin is called yrast. Usually the highly
excited nuclei quickly decay down towards the yrast band by the ‘statistical γ-rays’ and then travel down most of the distance to the ground state by discrete γ-rays of that, or another not too distant, band. Crossovers between different bands are possible but might be forbidden by parity or angular momenta considerations. Near the ground state there are usually many different levels of similar angular momenta and a less band dominated decay is possible.

In addition the nucleus may also vibrate adding another collective excitation mode. Transitions and levels associated with vibrations may be identified by sets of vibrational levels with predictable spacings emanating from the same rotational and particle excitation.

2.3 γγ-coincidence experiments

Two main experimental methods to produce excited nuclei of a certain species are fusion evaporation and Coulomb excitation. Both methods require a high energy beam that has a high concentration of a specific isotope. Both methods also require many γ-sensitive detectors covering as much solid angle as possible around the reaction region. Ideally the nucleus of interest should end up right in the middle of the detector set-up when it starts emitting discrete γ-rays.

2.3.1 Fusion evaporation

If the beam is directed at a suitable foil, some of the beam particles will collide with a target nucleus and, provided the energy of the beam is high enough to allow them to penetrate the Coulomb repulsion barrier, fuse with it. The resulting compound nucleus will be highly excited and quickly decay. An efficient way of emitting energy for such a nucleus is to start to evaporate particles. Neutrons and protons can take with them large amounts of energy if ejected from the core. Still after the evaporation has stopped, i.e. the excitation is not high enough to emit more nucleons, the nucleus will have much excitation energy to get rid of. Starting with statistical γ-rays the nucleus will usually soon find itself in a rotational band and start travelling down with discrete γ-rays.

The idea is to form compound nuclei that after the evaporation process have transmuted to the specific species one wants to measure. For example in several of the \( ^{61}X \) experiments described by Andersson [4] \(^{36}\)Ar and \(^{28}\)Si were fused into the compound nucleus \(^{64}\)Ge that evaporated into \(^{61}\)Cu, \(^{61}\)Zn or \(^{61}\)Ga by the \(3p, 2p1n\) and \(1p2n\) channels respectively. Which of the decay channels were used for each event can be identified by detecting the evaporated particles, separating the products coming out of the foil by standard magnetic and/or magnetic filters and by sending the recoils into an ionization chamber.

2.3.2 Coulomb excitation

In Coulomb excitations the beam and target are not fused but only inelastically scattered against each other. If the nuclear species of interest is stable one can choose to have either the beam or the target consist of that nuclei. In many modern experiments though, research of unstable exotic nuclei are of interest and then that species must be used as the beam. The other species can be whatever material suitable but since one wants to accomplish Coulomb excitations, materials with high atomic numbers is especially useful.
When the beam passes through the material and the nuclei come close to each other the intense Coulomb field will enable the kinetic energy to be transferred as it is converted to potential energy to excite either or both nuclei. Contrary to the fusion evaporation process this is done from the ground up rather than starting at an extremely excited state. Typically one reaches only the lower levels which can be a good thing since it is a way of knowing how far up in the scheme the observed transitions possibly can have emerged from. It does however complicate the intensity analysis since the lower states will be populated more frequently and some states might be excluded due to selection rules in the excitation procedure.

A great advantage of this method is that one knows what is happening in the scattering reaction since the electromagnetic interaction is known.

### 2.3.3 Detecting $\gamma$-rays

The ideal $2\pi$ coverage of the solid angle is usually hard to achieve due to things like the beam pipe and cables being in the way. What one would like is to cover a solid angle as large as possible with detectors having large energy precision in the energy range of the $\gamma$-rays (tenths of keV to some MeV). For that reason the preferred choice is often High Purity Germanium (HPGe) detectors. An example of such a detector is the GAMMASPHERE detector consisting of 110 different detectors arranged in a sphere which has $4\pi$ coverage if all detectors are in place [4].

To increase the spatial precision the Ge crystals may be segmented which basically means that an event in the crystal is read by several apparatuses at different positions. The position of the initial event can then be reconstructed based on the signals from each of the reading apparatuses. For high energy precision it is also necessary to make sure that only events where the full $\gamma$-energy is deposited in the same detector is used. This can be done by putting up a shield around the HPGe which detects scattered $\gamma$-rays trying to escape the crystal.

### 2.3.4 Coincidence matrices

The rapid $\gamma$-ray decays towards the ground state are so quick that one usually can not distinguish which $\gamma$-rays were sent out first and which ones came after. From the experimental set-up’s point of view all $\gamma$-rays from a single nucleus will be detected simultaneously. Whenever two different detectors in the experimental detector array detects two different particles simultaneously, one in each detector, the two particle events are said to be in coincidence. This means that the two transitions associated with the detected $\gamma$-rays must be part of the same decay chain.

One can easily extend the coincidence condition to three or more detectors. That would give increased information about the decay chains but at a cost of fewer coincidence events of that type and an increased difficulty for humans to get an overview of the resulting data. Assuming the coincidence condition is kept at only two $\gamma$-rays in coincidence one would, after the experiment, compile the event data into a two dimensional coincidence matrix. For each pair ($n$-tuple for higher coincidence dimensionalities) the events where both detectors have a signal is fed into the matrix at the indices given by the energy channels in each of the detectors. The matrix is then symmetrized so that it does not matter in which order the detectors in the pairs were chosen (effectively this means that only one half of the matrix actually contains any information). At each pair of indices in the matrix, corresponding to
an energy $A$ and an energy $B$, one will see how common it was for a $\gamma$-ray with energy $A$ to be detected at the same time as another detector detected a $\gamma$-ray with energy $B$.

There will typically be random coincidences caused by pure chance, due to other nuclei, but the true coincidences will form clear two dimensional peaks rising above the background. Each peak will cause a background though, due to Compton scattering. When the full energy of the $\gamma$-ray was not detected it will look like a $\gamma$-ray with smaller energy which naturally will coincide with the same other $\gamma$-rays as the full energy peak. This forms patterns emerging from each peak in the form of ridges in the $x$- or $y$-direction. When two of this ridges meet a false peak is created, why it is important to take ridges into account when doing background subtractions.
Chapter 3

Program details

3.1 The ROOT library

The ROOT\(^1\) data analysis framework developed by CERN is a very common tool in subatomic physics. It provides a platform independent library for many common tasks including graphical interfaces. It handles reading and writing of classes to and from files, browsing these files, manages histograms and functions including plotting and fitting, provides classes for many mathematical operations, provides classes for threading, etc. Simply, most things you will need writing a program for physics applications.

The ROOT code is released under the open-source license LGPL\(^2\). That together with an excellent documentation, including lots of examples and tutorials, easy to read code and fast high quality support on their forum makes ROOT available to most people even with limited programming experience.

Based on this, the choice of library for this thesis was rather clear. It also eliminated the need for any other libraries except the C++ standard template library.

The ROOT developers are experienced people who certainly have good arguments for their choice of design. Most choices have both benefits and drawbacks, but the users should make themselves aware of these choices and their consequences so that they can use ROOT in their own most efficient way.

Some class design choices can be a bit confusing. Some things are just an annoyance like the case when simple tasks as controlling the axis range of a histogram involve calling both methods in the histogram object as well as in the axis object owned by the histogram. Other things are questionable, like having to set the axis range of your two dimensional histogram to its extremum limits in order to be able to produce a projection of the full range.

The extensive use of global variables and registrations of instances of many of the most important ROOT classes are also of great concern. This feature enables code in completely different parts of your program to communicate but also to disturb each other if they, for example, happen to share the same name. In threading, the global registration may cause a thread to freeze when such an object is created. One could also argue that many global variables sometimes will encourage bad coding practices.

\(^1\)http://root.cern.ch
\(^2\)http://www.gnu.org/licenses/lgpl.html
CHAPTER 3. PROGRAM DETAILS

3.1.1 C++ interpreter

An important part of the framework is the scripting capabilities and the ROOT shell. Scripts in C++ can be run and classes and functions can be loaded runtime. This is all powered by the interpreter CINT and if more performance is needed ACliC provides a compiler interface for compiling the scripts. Naturally it is still possible to compile ROOT programs the usual way but scripts and interactive programs are useful complements.

ROOT also supplies a class for building an interactive shell into one’s programs which can add a lot of flexibility for advanced users. To make your own classes available to use in the interpreted environment all you have to do is to let ROOT generate a dictionary for them which contains all information ROOT needs about the class.

3.1.2 Signals

Signals are essential for most graphical interfaces in ROOT but are also useful in other circumstances. They are used to send information between an emitting object to a receiving object or function. Each time a certain event occurs in a class the signal implementation enables that event to trigger the execution of a function or method with appropriate arguments. In graphical interfaces it could be that a button is pressed which triggers the "Pressed()" signal to be sent out. The signal is then processed and information is relayed to any function or method listed as receivers which are then executed. But signals are by no means limited to graphical objects.

Any class can send out signals by inheriting from the ROOT class TObject or by including a macro in the class declaration. Either way the member method Emit() can be called from any other member method and any receiver can be registered by calling the member method Connect() in the signalling class.

All emitting classes must have their dictionary generated and in case of compiling the program, these dictionaries must be linked with the rest of the code.

A problem with the implementation is that signals should be disconnected upon the execution of either the emitter’s or receiver’s destructor. If that does not happen the memory associated with the connection object will be leaked in the best case. In the worst case the signal will trigger an execution of a method belonging to a deleted object i.e. that in many cases causes a segmentation fault. It has been found that the disconnection does not seem to work properly in some cases in at least some versions of the library (5.26 and 5.27 was tested) which calls for caution when using signals in complex ways and workarounds when they are not working as they should.

3.2 Interface and features

Two programs complementary to each other have been developed in this thesis. The first program called Fight for Fit! (FffFi) is a peak finding and fitting program for two dimensional coincidence matrices. The second called What a Coincidence!? (WhaC) is a scheme builder suited to import a set of fitted peaks exported from FffFi. Both should handle all platforms supported by ROOT but have only been tested on Linux.
3.2. INTERFACE AND FEATURES

3.2.1 Peak finder and fitter

**FifFi** provides a graphical interface (fig. 3.1) to view the matrix, fit peaks in one or two dimensions and start the background subtraction and peak finding routine. Optionally an interactive ROOT shell is also opened upon program initialization by stating this in a configuration file, enabling full control and scripting capabilities for the advanced user. The program’s purpose is to explore peak finding in two dimensions and provide WhaC with input data.

It is possible to view the original matrix, the background subtracted matrix, projections of these two matrices and the peak search matrix. Peak markers show the positions of found peaks and a unique identification number making orientation easier when viewing the data.

In addition to being able to fit the peaks in one (using either \(x\)- or \(y\)-projection) or two dimensions, nearby peaks can be grouped and fitted simultaneously. Chosen parameters can be fixed or locked so that no changes to them are possible. Locking is used after a successful fit to make sure the parameters are not changed but also to mark the fit as good. The program can be asked to provide estimates of the standard deviation of unfitted peaks based on previously locked peaks by assuming that the standard deviation is proportional to the square root of the energy.

The background subtraction and peak finding routines are described in detail in 4.1 and 4.2.

An automatic peak fitting procedure was also developed to remove some of the boring repetitive work associated with fitting a lot of peaks. The routine first sets up an area around the identified peak based on a parameter given by the user which defines when two peaks

![Figure 3.1: The program FifFi showing its tools for viewing the matrix as well as fitting peaks.](image)
are considered near each other. The data within the area is fitted to the function

\[ a = \left( \frac{x - p_1}{p_3} \right)^2, \quad b = \left( \frac{y - p_2}{p_4} \right)^2, \]

\[ f(x, y) = p_0 \exp \left( -\frac{1}{2} (a + b) \right) + p_5 \exp \left( -\frac{1}{2} a \right) + p_6 \exp \left( -\frac{1}{2} b \right) \] (3.1)

where \( p_i \) are the free parameters. A figure of merit is calculated in eq. (3.2) by investigating how big the difference is between the data in the histogram, \( h \), and the fit, \( f \), compared to the total data.

\[ X_{\text{FOM}} = \frac{\sum_{x,y} |h(x, y) - f(x, y)|}{\sum_{x,y} h(x, y)} \] (3.2)

If this figure of merit is too high a new area based on the standard deviations \( p_3 \) and \( p_4 \) are determined and the fit is instead tried in one dimension by projection of the data in \( x \)- or \( y \)-direction and fitting to the function

\[ f(x) = p_3 + p_4 x + p_0 \exp \left( -\frac{1}{2} \left( \frac{x - p_1}{p_2} \right)^2 \right) \]. (3.3)

The figure of merit is still determined by eq. (3.2) but in one dimension instead of two. If some peaks still could not be fitted with satisfying figures of merit the user is of course able to fit those peaks manually by changing the fit area and/or fixating parameters. The fit results can be viewed graphically within the interface as shown to the upper right in fig. 3.1.

### 3.2.2 Level scheme builder

**WhaC** also provides a graphical interface (fig. 3.2), configuration file parsing as well as an interactive ROOT shell. It allows the user to view the current level scheme and start the different scheme builder routines. Transition data can be supplied either as a C++ script in the interactive shell or as data imported from **FifFi**.

The scheme building routines are described in detail in 5.3 and 5.4.

### 3.2.3 I/O

Configuration of the program is done by creating a configuration file and specifying it on the command line. The configuration file allows the user to set some parameters used by different parts of the program. Mandatory parameters that must be set are those specifying how to read required files upon initialization, the rest of the parameters are optional, but recommended, to be set manually.

The background subtracted coincidence matrix, the resulting peak search matrix as well as the peak data can be saved by **FifFi** to a ROOT-file. Both **FifFi** and **WhaC** are able to load these files. The schemes constructed by **WhaC** can be exported to a number of graphical formats supported by ROOT.
3.3. CLASS STRUCTURE

Figure 3.2: The program WhaC after solving the scheme of a loaded set of transitions.

Figure 3.3: A simplified class diagram of the program suite developed. Only the main classes, main methods and main relations are shown.
3.3 Class structure

The programs have been designed with a graphical user interface in mind but without making classes depend on the graphical parts of the programs when it was not needed. The principal class structure is depicted in fig. 3.3. In the peak handling program FifFi the most important classes are Dataset, Peak and PeakGroup.

All peaks are instances of Peak and even if a peak is not grouped most managing like fitting etc. is made through the PeakGroup interface. The PeakGroup class provides a simple way of grouping the peaks and operating on groups of peaks. Only one instance of Dataset is usually needed and that object mainly keeps track of all Peak and PeakGroup objects but also the histograms and the automatic peak search and fitting routines.

The graphical interface of FifFi requires another set of classes including a main controlling window, HistView and ProjView for presenting the histograms, EditWindow giving the user control over the fitting of a Peak or a PeakGroup as well as presenting the fit results graphically and smaller classes of helper character. The different parts communicate through the main window class and signals.

Objects of the classes Peak and PeakGroup emit signals every time they are in some way changed, unless the signalling is shut off by explicitly calling the method holdSignals(). This communication assures that the information displayed is always updated and enables the user to edit the same data from different places in the program without risk of losing information.

In the scheme building program WhaC the class Dataset plays a much smaller role than in FifFi. It is only used if one constructs the transition objects from peak data saved in FifFi. The same goes for Peak; and PeakGroup is not used at all.

Instead the main parts are the classes LevelScheme, Level and Trans together with the namespace SchemeBuilder. Each instance of Trans represent a transition and owns pointers to an upper and a lower Level object. Correspondingly each instance of Level owns lists over the transitions feeding to and from the level. The Level and Trans classes are declared friends, that is e.g. deleting a Trans object feeding a level will set the Trans object’s lower level pointer to NULL as well as deleting the Trans object from the Level object’s list of feeds.

The namespace SchemeBuilder is almost only a collection of useful functions used by the level building routines. It does however also contain a public member, namely a list of unmerged schemes, if such schemes have appeared in the scheme building process.

The graphical interface of the program WhaC is a lot simpler than FifFi’s and mainly only one class, SchemeManager, is needed.

Both the programs’ main window classes are written as singletons which means that only one instance is ever allowed to exist at a specific time. Another singleton object is the instance of the class Options, a class holding information about all configurable options in the programs which is read from a configuration file upon initialization. Static methods allows retrieval of the information from anywhere in the program. If the Options instance is not initialized the main window classes will throw exceptions if attempts are made to create instances of them.

Interaction with most objects are possible through the ROOT interactive shell, via static functions giving access to the main objects, as well as through the graphical interfaces. All instances of Peak, Peakgroup, Trans and Level are assigned unique identification num-
bers which can be used as search keys to look them up in a collection like the Dataset or LevelScheme objects.

3.4 Known bugs

At the writing moment at least two bugs exist in the program. They are both related to the communication between the ROOT graphical layer and the X-server.

The first bug appears when viewing found peaks in the two dimensional histogram. The mouse-over text, revealing the unique peak identification number, is often shown at the wrong position if a two monitor screen layout is used. The bug has not yet appeared on any single monitor set-ups.

The second bug only affects some systems, possibly only some X.org\textsuperscript{3} versions. It was found that a NULL pointer was returned from the method GetBelowMouse in the ROOT class TGListTree, which normally should return an object pointer. This bug disables some mouse related features, like drag-and-drop. How and why the pointer became NULL instead is unfortunately not known.

\footnote{http://www.x.org}
Chapter 4

Finding peaks

As it is a common task, finding peaks in a spectrum, ROOT naturally has classes and methods for this ready to use like `TSpectrum2::SearchHighRes()`. In theory they should work well but it was found that they did not give satisfactory results. Especially background subtraction proved to be difficult and so a number of different approaches were tried in addition to the ones found in the ROOT package.

4.1 Background subtraction

As mentioned in 2.3.4 the coincidence matrix will in addition to random coincidences that form a general background also exhibit ridges. Removing these ridges cleverly simplifies the process of finding both large and small peaks greatly. However, background subtraction is not a problem with one single solution. Different backgrounds and situations call for different sub routines in order to produce a clean spectrum. A number of different methods were tried, the most interesting are summarized below, to find the optimal method for the two datasets this program was developed along with. The two somewhat different datasets demanded the background subtraction to be general and hopefully the resulting sub routine will manage to work well also on other datasets than the two the program was tested on.

4.1.1 Deconvolution

This is not a background subtraction algorithm but it is related since it is a method for identifying peaks. For the deconvolution algorithm to work one needs to know a response function i.e. the instrumental output from an input signal at a specific point $(x, y)$, usually assumed to be gaussian. The need to know the appropriate response function in advance is one of the drawbacks of this method. Knowing the output $y$ and the response matrix $h$, the input $x$ can be found by deconvolution i.e. one solves the equation $y = h x$. Due to noise in $y$ regular methods of solving the equation might turn out unstable. The algorithm, which is the same ROOT uses in part of `TSpectrum2::SearchHighRes()`, is described by Morháč et al. [5].
4.1.2 Peak clipping

This algorithm does it the other way around and actually removes the non-background rather than background. What is left of the spectrum after applying this algorithm is supposed to be the background. It is based on more simple principles and is much faster than doing a deconvolution.

This iterative algorithm’s two dimensional case uses eight points, the corners and mid-sides of a square, centred around a centre point being the bin currently treated. The eight points are used to estimate the local background by some simple averages and linear interpolations, taking eventual ridges into account, and then lower the centre value if the estimated background is found to be lower than the centre value. Each iteration increases the distance between the eight points and the centre point and will in that way be able to handle different widths of peaks and ridges. The algorithm described by Morháč et al. [6] is a multidimensional extension of the one dimensional algorithm SNIP described by Ryan et al. [7].

4.1.3 Median cut on sub ranges

After not being satisfied with the performance of the other algorithms on the two tested datasets this background subtracter was developed. Instead of clipping the peaks by lowering values above a certain locally calculated threshold this method, in a more conventional fashion, removes what is identified as background. Two main features of the coincidence matrix enables this method to work; the background is locally almost flat and the ridges are always running in the $x$- or $y$-direction.

The matrix is first traversed in the $x$-direction in blocks of $a$ for each $y$. I.e. segments of area $a \times 1$ is considered separately where $a$ is a predefined parameter. $a$ should be chosen significantly larger than a peak width but small enough that the background on the interval can be considered constant. All the bin values of the sub range is put in a vector and sorted. Thereafter the value at index floor($fa$) is picked, where $f$ is a predefined fraction $\in [0, 1]$ (if $f = 0.5$ the median is picked), and all values on the sub range is decreased with the picked value.

When the procedure has gone through the whole matrix it is repeated, but this time in the $y$-direction for each $x$. To produce a symmetric matrix the values on each sub range are actually not decreased immediately but rather decreased after both runs in different directions have been done.

The ridges are removed since they are locally constant in the direction they run and the routine goes in both the $x$- and $y$-direction. The method is very simple and crude but with reasonable parameters it gets the job done fairly well and only requires one pass. For more complicated irregular backgrounds this background subtracter would potentially fail miserably.

4.1.4 Pixel exterminator

Single pixels spread out over the remaining spectrum do usually not affect the computerized peak search very much but removing them can be an aid to the human eye. Therefore a last clean-up of the spectrum is done, after other background reducers have been run, by zeroing bins that have zero bins on all four sides. As long as the bin widths are significantly smaller than any other structure in the matrix it is most likely no problem to remove these bin counts.
4.2. PEAK SEARCH ALGORITHM

As a positive side effect the peak search algorithm described next will run faster the more zero bin counts the matrix has.

4.2 Peak search algorithm

Finding peaks is not just a matter of finding a bin count higher than all neighbouring bin counts since statistical fluctuations will cause noisy peaks and sometimes dispersed peaks. Smoothing the whole spectrum by letting each bin be influenced by a weighted average of the surrounding bins made it easier to find the peak bins but not significantly.

It turned out one should go the other way around, instead of smearing out the peaks one should concentrate them and let the high bin counts become even higher. Silagadze presented [8] the algorithm as ‘quantum mechanical’ since one interpretation and the inspiration to the algorithm is that the bin values are let to tunnel into nearby bins. The coincidence matrix itself has little to do with quantum mechanics. The method in two dimensions [9] redistributes a bin content, at indices \((i,j)\), to nearby bins, at \((k,l)\), weighted by the probabilities given by

\[
Q_{i+k,j+l} = \exp \left( \frac{N_{i+k,j+l} - N_{i,j}}{\sqrt{N_{i+k,j+l} + N_{i,j}}} \right) \quad (4.1)
\]

\[
P_{i+k,j+l} = \frac{Q_{i+k,j+l}}{\sum Q_{i+p,j+q}} \quad (4.2)
\]

where \(N\) denotes bin populations and \(0 \leq k, p \leq m, |l, q| \leq k\) controls how far away the distribution spreads. Running this over all bins in the matrix for a large enough number of iterations will create very sharp peaks due to the tendency to let local maxima grow larger on the expense of neighbouring bins. What is considered local is determined by \(m\).

These peaks are now simple to detect by e.g. looking for a vanishing first derivative or a sign change in the second derivative.

Since each state is generated by the last one, they form a Markov chain. Even though Markov chains often are used in conjunction with random numbers and probabilities there is no need for that in this algorithm. The bin population, \(N_{ij}\), is simply distributed according to the probabilities given by eq. (4.2) in a deterministic manner.
Chapter 5

Building a level scheme

5.1 Available information

The methods to build level schemes investigated in this thesis use only information obtainable from the coincidence matrix alone. Optionally one can supply an efficiency curve to make the program tweak the schemes using also intensity information. There are two main ways of supplying the needed information to the scheme constructing program WhaC.

1. Manually, preferably in a script, define all transitions and their coincidence relations to each other via the optional interactive ROOT shell.

2. Let the program automatically find the transitions by searching through a list of peaks originating from the peak searching program FifFi.

The first alternative may use the full power of C++ through the ROOT interpreter but requires pre-knowledge of the transitions’ coincidence relations. The second alternative requires that the FifFi program is used first to find and fit peaks in the coincidence matrix but the transitions and coincidences are found automatically by WhaC upon initialization. This is done by applying gates around the various peak energies and grouping peaks found inside the same gate into a transition and coincidence relations. It is of uttermost importance to remember that a peak in the coincidence matrix does not directly correspond to a transition but rather a coincidence between two transitions that have their energies given by the $x$- and $y$-coordinates. The program’s transition representation is constructed by taking all peaks with their $x$- or $y$-coordinate at a certain energy and add up the contributions.

In the end the program should have a list of transitions, each with an associated energy and energy uncertainty, and a matrix describing coincidences. No other information is needed nor wanted. Transition energies and coincidences represent ‘hard’ information that can be obtained in a very unambiguous fashion. No nuclear structure models are used anywhere in the program since that would introduce unnecessary bias into the level scheme building.

5.2 Previous level scheme solvers

Building level schemes manually can be a very tedious work so naturally previous attempts have been made creating a program for building level schemes automatically. In 1968 Helmer
and Bäcklin [10] based their program solely on transition energies and previously known levels. Their program was fed with level and transition data and in return it gave suggestions for new levels together with an estimation of how probable the suggestion was to get by pure chance. Some of the received new levels were then included in the ‘known’ set of levels in a typical semi-automatic manner and the program run once again.

Two years later Williams [11] wrote a program fitting transitions into a level scheme. That program was similar to the previous one, it used the energies of the transitions and nothing else except if the user already knew some levels. Still, he wrote that most of over 170 measured transitions could be fitted by his program into 32 levels providing great help in determining the level scheme of $^{148}$Sm. He also commented on the problem of only taking the energies into account; nothing in the program assured that the fitted transitions actually are allowed.

The same year also Johnson and Kennett [12] wrote a similar program but allowing for optional coincidence data to be supplied if available. They noted two things: That coincidence data was often missing for many of their transition pairs and that the amount of possible schemes was way too large for the computer power of that time.

The year after, in 1971, Rester [13] suggested a program that would be able to use additional information if available. The described algorithm was based on the energy differences of the transitions but additional information was concluded to be needed if unique complex schemes were to be found. For each run the user still had to judge what new levels to include in the next run but the possibility of extending the program to be completely automatic is mentioned.

Hons [14] described a program in 1978 that instead of beginning with the energy balance started with intensity and coincidences. The base for the scheme is made of transitions coincident with another transition of large intensity. Such cascades would later be combined if they had levels on the same energies and were reasonably supported by the coincidence information available. He noted though that most experimental coincidence data is not complete but that his algorithm still managed to agree with published schemes to a degree of 85-100%.

Five years later another program was written by Ditrői and Mahunka [15]. Their work primarily used the transition energies but intensities and selection rules came in as additional conditions. Their algorithm utilized the fact that the energy differences between two levels can be replicated by different pairs of transitions and the number of pairs replicating a certain energy difference can reveal information about the level ordering. With a few support levels they claimed that the method was very reliable.

In the following two sections this thesis explores two new algorithms for building level schemes. Both use methods distinguishable from the above suggestions in different ways.

### 5.3 Statistical approach

Monte Carlo simulations [16] are very general methods applied to many different kinds of problems throughout science. The variant used in this thesis is called simulated annealing and that is in turn a variant of a Monte Carlo Metropolis algorithm [17]. Its main advantage is that it has a tolerance against errors like undetected coincidences or even sometimes completely missing transitions.
5.3. **STATISTICAL APPROACH**

Even if something in the produced scheme seems contradictory it will not bother this algorithm much. Its main disadvantage is the same as its main advantage. Since the schemes produced by the algorithm sometimes contain logical errors they have to be examined carefully to see how much of it can be trusted. If there actually exists a complete solution to the level scheme the algorithm should be able to find it correctly, given enough iterations and a slow enough annealing.

5.3.1 **Metropolis**

The idea of a Metropolis algorithm is to sample a representative part of the, often huge, phase space in search of a global minimum. This is done by randomly going from one state to another and reward changes that lowers a, for the system defined, energy. The energy in this case is to a great extent arbitrary since the level scheme as such do not represent a real physical system, but it should in principle lower the energy when coincidences are obeyed and raise the energy when they are disobeyed.

If one applies the rather naive algorithm that accept all changes that lowers the energy but reject all other changes one gets what is called a local search. It will get trapped in a local minimum and the chances of finding the global one this way are very slim in a large complicated system, hence the name. What makes the Metropolis method useful is its ability to sometimes accept a new configuration even if the energy will become higher. If the energy becomes a lot higher the chances of acceptance is low but small increments can often be accepted and thus can local minima often be escaped.

Exactly what random distribution is used does not matter too much if the global energy minimum is the only thing of interest and no thermodynamic properties of the system is to be measured. The Boltzmann distribution is a common choice especially since it often describes real physical systems well. There are certain conditions needed to be fulfilled if the configurations in this iterative process, in the end of the simulation, will correspond to the Boltzmann distribution, \( p_B \). Let us denote each configuration with \( \sigma_i \) and the distributions \( p_i \) where \( i \) is the iteration number. The conditions are then:

- Each state \( \sigma_{i+1} \) depends solely on the state \( \sigma_i \). The states \( \{\sigma_i\}_{i=0}^{N} \) forms a Markov chain.
- The algorithm must provide the means of transforming any state into another. All states must be reachable from any other state. This is called that the algorithm is ergodic.
- Once the Boltzmann distribution is obtained it should not be left i.e. it is stationary. That is,

\[
p_n = p_B \Rightarrow p_i = p_B \quad \forall i > n.
\]

A common, rather simple, solution to the third condition is *detailed balance*. Let us denote the transition probability from \( \sigma \) to \( \sigma' \) as \( W_{\sigma \rightarrow \sigma'} \).

\[
W_{\sigma \rightarrow \sigma'} p_B(\sigma) = W_{\sigma' \rightarrow \sigma} p_B(\sigma') \Rightarrow \\
\sum_{\sigma'} W_{\sigma \rightarrow \sigma'} p_B(\sigma) = \sum_{\sigma'} W_{\sigma' \rightarrow \sigma} p_B(\sigma') \Leftrightarrow \\
p_B(\sigma) = \sum_{\sigma'} W_{\sigma' \rightarrow \sigma} p_B(\sigma')
\]
Where eq. (5.3) is equivalent to the previous eq. (5.1).

By factorizing $W$ into a proposal probability $F$ and an acceptance probability $A$ one can in turn ensure detailed balance by choosing $A$ appropriately:

$$A_{\sigma \rightarrow \sigma'} = \min \left( 1, \frac{F_{\sigma' \rightarrow \sigma} p_B(\sigma')}{F_{\sigma \rightarrow \sigma'} p_B(\sigma)} \right)$$

Assuming $A_{\sigma \rightarrow \sigma'} = 1$ eq. (5.2) becomes:

$$F_{\sigma \rightarrow \sigma'} p_B(\sigma) = F_{\sigma' \rightarrow \sigma} A_{\sigma' \rightarrow \sigma} p_B(\sigma')$$

$$F_{\sigma \rightarrow \sigma'} p_B(\sigma) = F_{\sigma' \rightarrow \sigma} F_{\sigma' \rightarrow \sigma'} p_B(\sigma') p_B(\sigma')$$

If all three conditions are satisfied the Metropolis algorithm is quite straight forward. A new state $\sigma_{i+1}$ is generated by changing state $\sigma_i$ in some way. If $\Delta E = \sigma_{i+1} - \sigma_i < 0$ the state $\sigma_{i+1}$ is accepted, otherwise the new state is accepted with a probability of $p_B(\sigma_{i+1}) = e^{-\Delta E / T}$ (Assuming $F_{\sigma \rightarrow \sigma'} = F_{\sigma' \rightarrow \sigma}$). If $\sigma_{i+1}$ is not accepted the old state $\sigma_i$ is taken as the new state $\sigma_{i+1}$.

The tendency to sometimes go up in energy is governed by the temperature. A high temperature means that high energy states also can be significantly populated. In the limit $T \rightarrow \infty$ all new states are accepted regardless of their energy and in the limit $T \rightarrow 0$ only states with lower or equal energy are accepted (i.e. a local search).

### 5.3.2 Simulated annealing

One method of helping the Metropolis algorithm finding the global minimum is to gradually lower the temperature during the simulation. In this thesis the temperature is just a variable controlling the acceptance and should not be confused with a real temperature. As the system anneals the algorithm tends to fall into deeper and deeper energy pits, having it harder and harder getting out of them. If the annealing is done slowly the chance of finding the global minimum should be a lot higher.

As mentioned before, as long as the minimum is the only important information one needs, it does not matter how you find it. But the annealing process will either disturb the detailed balance or make $A$ difficult to compute if other observables are of interest. Not considering temperature differences, we previously found that the acceptance depended on $e^{-\Delta E / T}$ but if $\sigma_i$ and $\sigma_{i+1}$ have different temperatures one finds that, with our choice of $A$, we need to know the partition function $Z(T)$ (Zustandssumme) before the simulation even is started.

$$A = \frac{p_B(\sigma_{i+1})}{p_B(\sigma_i)} = e^{-\left(\frac{E_{i+1}}{T_{i+1}} - \frac{E_i}{T_i}\right)} \frac{Z(T_i)}{Z(T_{i+1})}$$

The easiest way of solving this problem is to change the temperature slowly enough such that $Z(T_i) \approx Z(T_{i+1})$. Though, if thermodynamical properties are of interest one should probably not take any data contributing such variables right after a change in temperature. After a while the system will stabilize in the new temperature and obtain the correct distribution for the current temperature.
5.3.3 Statistical scheme building

Adopting the simulated annealing technique to the problem at hand, to build a level scheme out of loose transitions, requires two main implementations. First, an energy for the system must be defined. Secondly, the moves that will change one state into another must be chosen (and preferably accommodate the ergodic requirement).

Defining energy

The energy definition regulates what is considered good (low energy) or bad (high energy) in a scheme. Obeying coincidences should be the main good thing and false coincidences should be penalized. There are also good reasons for demanding that all the parts of the scheme are connected, otherwise it would in effect be two or more schemes. If one finds a state, such as the non-connected scheme just mentioned, which never should occur no matter how high the energy is, it can simply be assigned infinite energy and thus never be accepted. In most schemes transitions are found in a tree structure emanating from the ground level; it could help rewarding schemes that have such a tree structure by penalizing each transition that cannot be found in a direct chain connected to the ground state.

Since all transitions have an associated energy uncertainty there is a possibility that the different transitions’ energies and the energies of the levels they are attached to have small mismatches. To find the best match an energy penalty is assigned to each transition based on the energy difference between the higher and lower levels and the transition energy, $|E_{\text{high}} - E_{\text{low}} - E_t|$.

If the transition data includes useful intensities one could try to define an additional energy contribution to encourage intensity balance on each level. However, because of side feeding it is not always the case that the intensity fed into a level is the same as the intensity fed out from the level. A similar problem is that there is no obvious way of determine the relative sizes of the contributions. Each contribution is therefore at runtime scaled with a configurable constant. The higher the value of that constant the more important that energy contribution will be. If one so wishes the constant for a particular contribution can be set to zero to disable that contribution completely. Negative values are in theory possible but will result in the program trying to build bad schemes.

In the end the energy was defined by the following rules:

- If not all transitions are connected into one scheme the energy is infinite.
- For each transition not reachable in a direct chain structure from the ground state the energy is increased by $C_{\text{conn}}$.
- For each obeyed coincidence the energy is decreased by $C_{\text{coinc}}$.
- For each false coincidence the energy is increased by $C_{\text{coinc}}$.
- For each transition the energy is increased by $|E_{\text{high}} - E_{\text{low}} - E_t| \times C_{\delta E}$.
- For each level the energy is increased by $|\sum I_{\text{in}} - \sum I_{\text{out}}| \times C_{\text{int}}$ where $I$ denotes intensity.
Defining moves

How the next configuration is generated can be more or less arbitrary as long as one makes sure that the phase space is explored properly. In the program WhaC in this thesis each new configuration is generated by making moves of the elements in the previous scheme. In each iteration one of the move types is chosen at random and executed. To assure ergodic behaviour transition should be able to move rather freely and levels need to be able to move and merge with each other. The following moves have been implemented:

- A transition is detached from the scheme and reattached randomly.
- A whole branch is detached from the scheme and reattached randomly.
- A level changes its energy by $\pm \delta$.
- Two adjacent transitions swap places.
- A level searches for another level with similar energy and merges.

The branch move might seem superfluous as it could be accomplished by several single transition moves but such a move can be efficient since a whole structure is moved at once. Moving several single transitions often requires passing an energy barrier and that may take several attempts before it succeeds and thus increasing the number of iterations the program needs to search the phase space properly.

In fact it was previously assumed that the proposal probabilities $F_{\sigma \rightarrow \sigma'}$ and $F_{\sigma' \rightarrow \sigma}$ was equal which is not always the case for the branch move. However, since the algorithm only searches for a minimum it does not matter how it is found and this small discrepancy can be tolerated.

Annealing

The annealing is rather simple. The initial temperature usually is set quite high allowing the routine to explore the phase space rather unrestricted. After a predetermined number of iterations the temperature is lowered by multiplying it with a constant lower than one. A stop temperature greater than zero should be set low enough such that the system stabilizes
5.4 Logical approach

This thesis presents yet another approach differing from all the previously mentioned programs but most closely resembles the work presented by Hons [14]. Instead of starting with the transition energies it is driven by obeying the coincidence data completely. The algorithm will only produce schemes which are completely correct in the sense that all coincidences between the included transitions are satisfied and no others. Failure to do so results in the program presenting sub schemes of consistent parts of the scheme.

The goal of this approach is to create a logically consistent level scheme based upon previously found transitions and coincidences, derived from peaks in the coincidence matrix or manually input. The method, described in detail below, is based upon putting multiple coincidence conditions on a set of transitions until a small group has been singled out. This small group will then have a trivial structure. After finding several such small sub schemes they will be merged into each other, keeping track of possible permutations of transitions,
and, if no information is missing or erroneous, produce one large scheme containing all transitions.

Given an ideal case, where all coincidence data supplied to the algorithm is correct and complete, the algorithm guarantees a correct, complete resulting level scheme. If the coincidence data in any way is inconsistent such that a total scheme is impossible to produce the algorithm will try to make the best of the situation and display the sub schemes, that in the end could not be merged into each other, to the user.

The whole routine \texttt{buildScheme} can be followed in the flowcharts depicted in fig. 5.3, 5.6, 5.8 and 5.7. A key to the symbols used is found in fig. 5.1.

The only mandatory argument to \texttt{buildScheme} is the list of all transitions one wants to build the scheme out of. The routine is recursive and will therefore call upon itself with different subsets of the initial list. However the routine works the same way whatever recursive level it is currently working on. Each time it is called it will return a scheme of the transitions it was called with.

\subsection{5.4.1 Finding something to start with}

The first thing the routine does is to check whether all the transitions in a list supplied are in coincidence with each other. If so the corresponding scheme of those transitions can be
constructed by merely stacking the transitions on top of each other. In that way each and every transition will be in coincidence with the others. This is the start of all schemes and when these small almost trivial sub schemes have been found they will later on be merged into larger schemes with more complicated structures.

In most cases there are some transitions in the list that are not in coincidence with all others. The routine will then pick the first transition in this list (but it could just as well be picked at random) and search for transitions in the list that are coincident with the picked main transition. Once that is done the routine recursively calls upon itself with the new list of coincident transitions and receives a new scheme in return. Sometimes the list will be empty and no recursive call is needed, the new scheme will consist of only the main transition.

If the recursive call was made, the newly constructed scheme will be combined with the main transition. Since the main transition is coincident with the whole scheme there should be one level in the scheme that can see all the transitions. At that level the main transition is inserted. The only thing left to do with the scheme is to search for possible permutations of the scheme (see 5.4.2).

All the transitions involved in creating the last scheme is marked as ‘handled’ and as long as the list contains ‘unhandled’ transitions the procedure is repeated. When all transitions are ‘handled’ the routine has gathered a list of sub schemes that now have to be merged.

### 5.4.2 Permuting schemes

Finding possible permutations is an essential step to make the routine work correctly. We will define these permutations as a group of ‘permutables’ that are in coincidence with each other and share all their other coincidences. A ‘permutable’ is a single transition or a group of transitions having properties that allow them to permute i.e. swap places with each other.

The most simple example is two transitions on top of each other, no other transition is attached to the middle level. Using only coincidence information the schemes where transition $A$ is on top of transition $B$ is equivalent to the scheme where $B$ is on top of $A$. In this simple example only two different configurations are possible but each time you add a ‘permutable’ to the permutation group the number of possibilities rises to $N!$, if $N$ is the current number of ‘permutables’ in the group.

When it comes to ‘permutables’ being groups of transitions the picture gets a bit more complicated, but not that much. The ‘group-permutable’ consists of a number of transitions trapped behind two different levels. No non-member transition may be found between these levels and all transitions must, in one way or the other, form chains between the two levels. The most simple example of such a group is the cross-over, where subsequent transitions $T_1$ and $T_2$ going from level $L_0$ to $L_2$ via $L_1$ gets crossed-over by transition $T_0$ taking the shortcut connecting level $L_0$ and $L_2$ directly.

It becomes quite possible for a scheme to have rather complicated permutations since there can be permutation groups within ‘group-permutables’. However, it is impossible for two permutation groups to overlap unless one is contained in the other. In fig. 5.4 two permutation groups are shown, a larger one in which one of the ‘permutables’ is a group and a smaller one contained within this ‘group-permutable’.

A second kind of permutation exists although permutation might not be the best word for it. In some cases it is possible to identify groups of transitions that have to be turned upside down, i.e. flipped, in a scheme. Those groups are in some way similar to the ‘group-permutables’ since they also consist of transitions forming chains between two levels but
CHAPTER 5. BUILDING A LEVEL SCHEME

Figure 5.4: The 100, 55 and 45 keV transitions form a ‘group-permutable’ that together with the 30 keV transition (a regular ‘permutable’) form a permutation group. The 55 and 45 keV transitions form a smaller permutation group contained within the larger one.

Figure 5.5: All transitions except the 100 keV transition are part of the ‘flippable’. Due to the 25 keV transition in the middle no regular permutations are possible.

there may exist non group members also attached to the limiting levels. Still, the condition of not changing any coincidences upon a flip applies. Additionally there is no need of flipping a group of transitions if the flip can be effectively performed by regular permutations. These cases can be sorted out by investigating whether a level that sees all group members exists within the group or not.

An example of a ‘flippable’ is depicted in fig. 5.5. No permutation, as previously described, can achieve the same effect as the flip. All different schemes one can accomplish using these permutation techniques are equivalent from a coincidence point of view. When
5.4. LOGICAL APPROACH

merging schemes these permutations are essential since the program must find permutations of the schemes that make them fit together and in the same time make sure not to create schemes incompatible with the coincidence information given.

5.4.3 Building larger schemes

The first thing to do when trying to merge the sub schemes into one larger scheme is to, for each pair, check whether there exists a non-empty subset of transitions that are present in both schemes. If so, it might be possible to merge them, but a merge is only guaranteed in an ideal case. In either case it might be that some other merge needs to be done first and one has to skip the merge of this particular pair for a moment. How to try and merge two schemes is described in detail in 5.4.4. In the unfortunate case when it is impossible to merge any schemes, even though they share transitions, one of the schemes is put aside as ‘unmergable’. This also indicates that a complete consistent solution to the level scheme can not be found and that the experimental data is incomplete or incorrect.

When all possible merges are done, all schemes left will have no transitions in common and are therefore parallel structures in the level scheme. These are handled by identifying a ground state for each sub scheme and merging all ground states into one level. The remaining scheme should now, if no ‘unmergable’ has shown up, contain all the transitions the routine initially was supplied with and the only thing left to do, before returning the

Figure 5.6: Checking for scheme overlaps in the buildScheme sub routine. Continuation of the flowchart in fig. 5.3. The flowchart continues in fig. 5.7 and the merge sub routine can be seen in fig. 5.8.
5.4.4 Merging two schemes

If two schemes share at least one transition it might be possible to merge them into one scheme. A flowchart of the process is depicted in fig. 5.8.

Two checks are made to make sure all coincidences are obeyed. The first check is making sure that it is worth trying to merge the schemes at all. All shared transitions’ upper and lower level energies are compared between the two schemes. If the schemes share a transition it means that both schemes contain a clone of the same transition. The only difference in energy, between for example the energy of a specific transitions’ higher level in one scheme and its counterpart in the other scheme, allowed is a translation constant which must be the same for all pairs of counterparts. If this condition is not fulfilled no merge will take place since the schemes are obviously not compatible.

Some schemes can be made compatible by utilizing the freedom of permuting and flipping the schemes (see 5.4.2). All possible permutations in both of the two merge candidates are tried before giving up and trying a new pair of schemes.

When, in one way or the other, the schemes have been made compatible a ‘merge-scheme’ will be produced. The base of this new scheme is a clone of the largest of the ‘to-be-merged’ schemes. For each transition the schemes have in common, the sub routine createMerge finds the corresponding transition in the non-base scheme. It then searches for transitions directly connected to the common transition that are not yet a part of the base scheme.

After adding any newly found transitions in such a way that they retain the same relationship with the common transition they had in the other scheme, the routine adds the new transition to the end of the list of shared transitions. That means that the shared transition and the ‘to-be-added’ transition shares either the upper or lower level and will be either par-
5.4. LOGICAL APPROACH

Figure 5.8: Merging sub schemes in the **buildScheme** sub routine. A sidetrack to the flowchart in fig. 5.6

...allel or in coincidence depending on how the situation looked like in the non-base scheme. Since all transitions within a scheme are somehow connected the resulting scheme will, after looping over all transitions in the list of common transitions, contain the union of the two initial schemes’ transition lists.

Often there will be a need for merging some of the levels in the ‘merge-scheme’ so the routine **conditionalLevelMerge** is called (see 5.4.5). The second check is now performed which will determine if the merge is accepted or rejected. If all coincidences are fulfilled and no false coincidences have appeared, the new scheme will replace the two initial schemes and be searched for permutation possibilities, before the routine returns to merge the rest of the schemes.
CHAPTER 5. BUILDING A LEVEL SCHEME

5.4.5 Merging levels with coincidence conditions

Mainly due to two important reasons there exists a sub routine for merging levels in the program. It will merge levels of approximately the same energies, providing coincidences in some way support it. Each time two schemes are merged additional coincidence conditions must be fulfilled. Levels that previously were in different schemes and even at different energies will perhaps after a merge find themselves side by side in the new larger scheme. Often, but not always, these levels will be found to actually be two versions of the same single level and have to be merged to be able to fulfil the new coincidence requirements. To fulfil these requirements is the first reason of conditionally merging levels after each successful scheme merge.

The second reason is of a more technical character. The implementation of grouping transitions as ‘permutables’ or ‘flippables’ are dependent on having a lower and higher level. The code will not correctly identify these groups unless the levels merge before searching for permutation possibilities. Having the same energy is however not enough to decide that two or more levels should be merged, it is of course possible for a level scheme to have two separate levels of the same energy but with for example different spin and parity.

Hence, at least one of the following conditions must be fulfilled before the levels are merged.

![Figure 5.9: Flowchart of the createMerge sub routine](image-url)
1. A transition feeding the first level is coincident with a transition decaying from the second level. Or vice versa.

2. Both levels lack transitions feeding them but share a level somewhere in their decay chains.

3. Both levels lack transitions decaying from them but shares a level somewhere in their feed chains.

The first rule is rather obvious since it only states that coincidences must be obeyed. The two other rules are needed for the ‘permutable’ and ‘flippable’ search. Out of context of the rest of the algorithm one might think that one would sometimes accidentally merge two levels that should be kept separate due to the two latter rules but this is not the case. Since the algorithm always works with subsets of transitions that all share one coincidence the constructed sub schemes will always contain full branches of the complete scheme. Thus, the program will never reach a situation where a new transition would be placed on top of a level that is the result of following rule 2. The resulting structure may however move, if permutations allow it to, but that can only happen if the levels are correctly merged, so that the ‘permutables’ can be correctly found.

There is one single situation where the level merge might merge levels by mistake. It is when two levels lack (detectable) transitions feeding them, so that they may be merged according to rule 2, but have in reality different spin and/or parity. From a logical point of view though, the state where the levels are merged are just as correct as the state where they are not since the algorithm only uses the supplied coincidence information which in this case was flawed.

### 5.4.6 Worked out example

Getting an overview of the algorithm from a pure textual description is not necessarily easy. What follows here is an example in which a small scheme of seven transitions is worked through and solved by the logical algorithm. Devoted readers are encouraged to follow the example on a piece of paper keeping track of how schemes are created and merged. The complete experimental information needed is found in table 5.1.
CHAPTER 5. BUILDING A LEVEL SCHEME

Let \( i = 0 \)

Is \( i < \) number of levels in \( LS \)?

Let \( L_1 \) be the \( i \)'th level in \( LS \)

Are there other levels in \( LS \) with the same energy as \( L_1 \)?

Let \( i = i + 1 \)

Let \( L_2 \) be the first other level with the same energy as \( L_1 \)

Does \( L_1 \) have a decay that is coincident with a \( T \) feeding \( L_2 \)?

Does \( L_2 \) have a decay that is coincident with a \( T \) feeding \( L_1 \)?

Merge \( L_2 \) into \( L_1 \) and remove \( L_2 \) from \( LS \)

End

No

Yes

No

Yes

Yes

No

Yes

No

Yes

No

Yes

No

Yes

Figure 5.10: Flowchart of the \texttt{conditionalLevelMerge} sub routine

The algorithm is started by calling the recursive function \texttt{buildScheme} with the full list of transitions (\( A-G \), see table 5.1). Since they are not all in coincidence with each other and none of them are yet handled, any of them could be picked as the first main transition. Let us pick \( A \) as the main transition and make the first recursive call with a list of the transitions that are coincident with \( A \).

The function \texttt{buildScheme} is thereby called with the transition list: \( C, F, G \). All can be seen to be coincident with \( A \) in table 5.1. They are still not all coincident with each other so another main transition is picked: \( C \). Neither \( F \) nor \( G \) are coincident with \( C \) so \( C \) is for the moment put aside in its own sub scheme. On this recursive level \( C \) has been handled and therefore a new main transition is picked from the unhandled transitions \( F \) and \( G \). Assume \( F \) is picked and therefore the next recursive call will be made with a list containing only \( G \).

\( G \) is immediately put in its own scheme which is returned to be combined with this scheme’s main transition, \( F \). The result is a scheme containing \( F \) on top of \( G \) which now should be merged with the previous scheme containing only \( C \). Since the schemes do not share any transitions they must be parallel and the merged scheme will consist of a ground level fed by \( C \) and \( G \), \( F \) still on top of \( G \). This merged scheme is then returned to the first
5.4. LOGICAL APPROACH

The only possible placement of $A$ is as a decay from the lowest level in the returned scheme since $A$ will then be in coincidence with all the transitions; $C$, $F$, and $G$; in the scheme. As a result of adding $A$ to the scheme, the top levels directly above $C$ and $F$ respectively can now be merged since they are, within errors, at the same energy and share at least one transition in their decay chains (see 5.4.5). This is the first scheme returned to the first recursion level.

Back at the first recursive level the transitions $A$, $C$, $F$ and $G$ are now handled. Let us pick $B$, which is in coincidence with $D$ and $G$, as the next main transition and call the function once again with the new coincidence list. The function returns a scheme consisting of $D$ on top of $G$ since they are in coincidence. The main transition $B$ is then simply added to the chain so that the complete sub scheme is $D$ on top of $G$ on top of $B$. This is the second scheme returned to the first recursion level.

The only unhandled transition from the original list is now $E$, coincident with $C$, $D$, $F$ and $G$. In the same fashion as before, the choice of $C$ as the next main transition results in a scheme containing $D$ on top of $C$. After that the unhandled transitions on this recursive level are $F$ and $G$. Taking $F$ coincident with $D$ and $G$ in the current list as main transition results in a scheme containing $D$ on top of $G$ to which $F$ is added under $G$.

Another scheme merge is called for but this time the two merge candidates actually share a transition, $D$. Starting the with the largest scheme; the one with $D$, $F$ and $G$; $C$ is added from the smaller scheme under $D$ since $C$ is placed that way in the smaller scheme. A check is made confirming that all coincidences are obeyed and after that the scheme is returned to the first recursion level and is combined with the main transition $E$. Since $E$ should be in coincidence with the rest of the scheme it can be placed e.g. on top of $D$. Of similar reason as with the first scheme returned to the first recursion level the levels beneath $C$ and $F$ can be merged in this third scheme.

The final merges can now take place. The first and second schemes share the transition $G$ and have four ($2 \times 2$) respectively six ($3!$) possible permutations that have to be tried before giving up merging them. Unfortunately none of the resulting merged schemes can accommodate the coincidences correctly no matter how the schemes are permuted, we will see why when the scheme is complete.

Instead a merge of the first and third scheme is tried. They share the cross-over structure $C$, $F$, $G$ but the non-shared transitions are in anti-coincidence and must be put in two parallel branches. The largest scheme (the third) has twelve ($2 \times 6!$) possible permutations while the first still only has four. If the whole first scheme is turned upside down the both schemes can be merged and the resulting scheme will consist of the cross-over structure $A$, $D$, $E$ on top of the other cross-over $C$, $F$, $G$. The top-most levels are merged into one since the energies of $D$ and $E$ equals, within errors, the energy of $A$ and there exists a shared transition in their decay chains.

The second scheme can now be merged but first both the schemes have to be permuted so that they fit together. In the big scheme $E$ and $D$ need to switch places as well as $G$ and $F$. The second scheme must be transformed into the structure $D$ on top of $B$ on top of $G$. The transition $B$ will now fit nicely between $D$ and $G$ in both schemes, thus the resulting merge can be accepted.

All sub schemes are now merged and all coincidence relations are obeyed. The resulting example scheme is actually the $^{181}$Ta scheme found in fig. 6.6 which was found during the investigation of the $^{170}$Er scheme. One now sees why the first attempted merge on the first
recursive level failed. To accommodate the coincidence relation between $D$ and $F$ one must first have $E$.

5.5 Distinguishing up from down

If the experiment provided useful intensity information, i.e. there exists a reasonably correct efficiency curve and the experimental method used was not Coulomb excitation but fusion evaporation, the program can after solving the scheme by coincidences try to guess what is up and what is down in the scheme. By mere coincidence logic this is not possible.

For each level with at least one incoming and one outgoing transition, the incoming intensity is subtracted and the outgoing added to form a figure of merit. Because of side-feeding one generally expects the outgoing intensity to be equal or greater than the incoming. The scheme is placed in the state of highest figure of merit. The investigated states do not only determine up and down for the whole scheme but also for any remaining permutations. That makes it possible to also determine the order within cascades if that has not yet been determined by coincidences.
Chapter 6

Results

6.1 Background handling

6.1.1 Deconvolution

The deconvolution peak search implementation, available in the class TSpectrum2 as the method SearchHighRes, was found to identify many peaks but also had a tendency to miss some and misidentify ridge crossings as peaks. Also some problems where the routine ‘dragged’ peaks apart and reported one peak as two arose. The problems seem to mainly lie in the fact that the peak widths continuously vary along both axes in the histogram. The deconvolution response function must take that into account to work properly.

Also the deconvolution was extremely slow even for quite few iterations and very memory devouring since several copies of the matrix is made during the run of the routine.

6.1.2 Peak clipping

This algorithm turned out to be very parameter dependent. The parameter determining both the amount of iterations and the width of the clipping points influenced the results greatly. Chosen too low almost no clipping occurred at all but chosen too high all of the histogram was clipped away. Small peaks were not spared and the performance at large was uneven, one parameter value might be good for one part of the histogram but bad for another.

6.1.3 Median cut on sub ranges

The simple previously described algorithm developed to remove the background can be seen in action in fig. 6.1. The ridges disappear, the peaks on them are still there and even very small peaks are mostly spared. If one chooses to use the median or some other ‘adjusted’ median depends on how the histogram looks, but values slightly above 0.5 seemed to give good results.

6.2 Peak finder algorithm parameters

The peak finder algorithm based on Markov chains was tested with different bin windows and numbers of iterations. The bin window, determining how far away bins can influence another, gave best results when it was chosen quite small. The number of peaks correctly
identified without too many false positives was highest for a bin window of two or three. This is good since the number of calculations quickly rises when the window gets larger.

Given the bin window value above, the number of iterations needed was found to be about 20. Going further with more iterations gave no improvement on the results. A modern computer completes the peak searching within some minutes for a $4096 \times 4096$ matrix.

A test was also done trying the extremes. A bin window of 30 was tried for 100 iterations but did not seem to improve anything at all and took quite a long time to finish; the test computer worked on it for more than a full day.

### 6.3 Level scheme builder algorithms

#### 6.3.1 Perfect data

To test if the algorithms found the right solution to an ideal case a constructed test scheme was created. It is depicted in fig. 6.2 and is the same scheme of the lower $^{170}\text{Er}$ levels as discussed by DiJulio et al. [18].

**Statistical approach**

Given enough iterations, high enough temperature and low enough annealing rate the metropolis routine recreates the correct scheme in fig. 6.2. It does take some time (in the order of minutes) if one wants to be sure the right scheme is found.

**Logical approach**

The logic algorithm was designed primarily for constructing schemes with complete coincidence data. Using the constructed set of transitions and related coincidence information the correct scheme was found every time no matter which order the transitions happened to be in. The only uncertainties left when the final scheme has been found were the trivial ones.
6.3. LEVEL SCHEME BUILDER ALGORITHMS

The peaks experimentally found from $^{170}$Er were not all the peaks one can see in the constructed scheme in fig. 6.2. Much of this was due to low efficiency for detecting low energy transitions, especially since there are many coincidences with the 79 keV transition in this scheme. Some transitions identified as a parallel structure associated with $^{181}$Ta was removed from the set prior to these tests. What follows is how the different methods reacted on the incomplete set of information.

**Statistical approach**

The Metropolis routine did pretty well considering that so many coincidences were missing. In fig. 6.3 one can find a system of 20 transitions in the same relative positions as in the constructed ‘true’ scheme. The deviating transitions are the ones of 482, 855, 646, 932, 934, 789, 78 and 569 keV. A total of 76 coincidences of 84 possible were met and only 10 false coincidences were produced.
Figure 6.3: The resulting scheme from running the metropolis routine on experimental coincidence data.

Figure 6.4: 10 000 moves were performed for each data point. Starting on 1 000 the temperature was decreased by 1% after each set of moves until it reached 0.1.

Depicted in fig. 6.4 is the annealing process for a rather thorough run. The energy can be seen to vary a lot as long as the temperature is high but stabilizes as it gets lower. Below a
temperature of about five the system is more or less frozen.

Logical approach

The extracted coincidence information was contradictory and the logic scheme solver had to split the scheme into several parts (fig. 6.5).

Several decay chains from fig. 6.2 can also be found in fig. 6.5 although the energies do not match exactly due to some calibration uncertainties. In the following paragraphs the energies in fig. 6.2 will be used.

In fig. 6.5(a) the three transitions are indeed in the same chain; the order is a mere permutation issue. There should be two more transitions in that chain but they are missing due to the missing coincidence information. That is also the reason this scheme has not been merged with the others.

In the second scheme (fig. 6.5(b)) the transitions of 374 and 280 keV have swapped places. The reason is that the 1163 and 694 keV transitions are missing the coincidences with the 181 and 78 keV transitions and must be placed under the 280 keV transition instead of above as
CHAPTER 6. RESULTS

in fig. 6.2. The two transitions of 567 and 646 keV are missing on top of the 931 respectively 855 keV transitions, otherwise the whole decay chain would have been present.

The same pair of transitions are also missing in fig. 6.5(c) which as for the rest depicts a correct scheme involving 17 of the 28 experimentally seen $^{170}$Er transitions. However, the small 78 keV transition has disappeared due to efficiency problems at low energies.

The last scheme depicted in fig. 6.5(d) includes two parallel schemes of which one is turned upside down. Between the 181 and 567 keV transitions a 750 keV transition should be present but once again the data supporting such placement is missing. Even though many of the 78 keV coincidences are missing one sees that the second level in fig. 6.5(d) is still at the correct energy.

The mentioned and removed $^{181}$Ta transitions found together with the $^{170}$Er peaks was tested separately. Fig. 6.6 depict the solved lower levels of $^{181}$Ta. The many crossovers in this small scheme are seen to pose no problems for the scheme solver routine.

6.3.3 Speed

To test how much time is needed to get results from the methods some benchmarking was performed on a normal PC. All tests were made on a single, otherwise unloaded, AMD Phenom core running at 1.05 GHz. Comparison of the real time and the CPU time shows that the programs bottleneck is the CPU and nothing else.

The Metropolis builder can take quite some time to run and that time will vary with how often one makes a certain kind of move and how thorough one like to explore the phase space. Usually less than a minute is enough to solve a smaller scheme but with more transitions you will need more time.

The logic scheme solver solves small schemes very fast rather independent on which order the transitions are handled in. An ideal case with 34 transitions is solved in 0.03 seconds.
on average and a non ideal case with 28 transitions is done in roughly the same amount of time.

Larger schemes can take more time since the possible permutations easily can reach several millions. More inconsistent information leads to a larger possibility of getting stuck with millions of attempts to merge two schemes that are in fact impossible to merge. The time for such schemes varies a lot. The test subject, a scheme with 79 transitions with inconsistent coincidence data can take up to five minutes on the same machine the previous test was made on but it can also take only 0.3 seconds if the transition order is favourable.

Since the program has been in constant development there exists code that makes redundant checks and provides the developer with debug information. Removing these parts of the code would improve the running speed of the program noticeable.
Chapter 7

Conclusion and outlook

Methods of peak finding and fitting have been evaluated and two methods of automatically constructing level schemes have been developed and tested.

The background subtraction investigation implies that a single method seldom is good enough for all schemes or even all parts of a single scheme. The simple algorithm developed *Median cuts on sub ranges* provided the best background subtraction among the tested algorithms but tests using other schemes might find the opposite. It did show however, that a background subtraction method does not necessarily need to be very complicated.

A method that was not tested, but may be fruitful, is calculating the Compton trail after a found and fitted peak and removing it from the coincidence matrix as background. That would directly identify and eliminate the often troublesome ridges.

The implemented peak search routine demonstrated a good way of finding peaks in two dimensions and the principle could be extended to more dimensions if needed. Although fitting did not work as well in two dimensions as in one, finding peaks is much easier since a second dimension in the histogram means another dimension to separate the peaks in. This possibility to separate the peaks would be even more emphasized in three or more dimensions and it would probably be a good thing to include viewing modes of any dimensionality (as long as it would be human comprehensible) in most programs dealing with γ-ray coincidence matrices.

The simulated annealing routine did manage to produce rather good results. Together with program output of which coincidences are correct and which are not, it could serve as a starting point for building the complete scheme. The routine could easily be extended by new kinds of moves to create new configurations, possibly making the routine more efficient.

Most previous attempts to automatically construct level schemes have used the transition energies as their starting points [10–13, 15] but there exists at least one other attempt to use the coincidences as the primarily source of information [14]. This thesis has shown that it is possible to create a level scheme solver based mainly on the coincidence relations between the detected transitions without using anything but those relations and the transition energies. On top of that the developed logical scheme solver always produces ‘coincidence-wise’ correct results, as long as the information supplied to the algorithm is correct.

Not much work seems to have been done on these problems since the eighties but now, with much faster and capable computers, it is possible for level scheme solvers, like the logical one in this thesis, to work through all possible permutations, usually in a matter of seconds, if needed. The logical method presented here is a starting point, showing that it is
possible to completely solve ideal cases and provide help when solving cases with missing or faulty information. Possible improvements could include e.g. methods of distinguishing doublet transitions from the coincidence matrix or a better semi-automatic user interface letting the user manually modify the schemes when a non-ideal case has caused the algorithm to fail to create a single scheme.

Due to the lack of attention and the improvements of computer hardware it is reasonable to once again start looking for more solutions of automatic character. Not because the existing programs are insufficient but since it might be possible to extend them with additional functionality of the kind this thesis has investigated. That way both time and numerous human errors would be spared.
Bibliography


